I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: December 25, 2016

S. Hares
Huawei
S. Kini
Ericsson
L. Dunbar
Huawei
R. Krishnan
Dell
D. Bogdanovic
Juniper Networks
R. White
Linkedin

June 23, 2016

Filter-Based RIB Data Model
draft-ietf-i2rs-fb-rib-data-model-00

Abstract

This document defines a data model to support the Filter-based Routing Information Base (RIB) Yang data models for I2RS. A routing system uses the Filter-based RIB to program FIB entries that process incoming packets by matching on multiple fields within the packet and then performing a specified action on it. The FB-RIB can also specify an action to forward the packet according to the FIB entries programmed using the RIBs of its routing instance.

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 December 25, 2016.

Table of Contents

1. Introduction

The Interface to the Routing System (I2RS) [I-D.ietf-i2rs-architecture] architecture provides dynamic read and write access to the information and state within the routing elements. The I2RS client interacts with the I2RS agent in one or more network routing systems.

This document provides a yang module for the I2RS filter Based Routing Information Base (FB-RIB) and describes the I2RS interaction with routing filters within a routing element. The informational model for the FB-RIB is in [I-D.kini-i2rs-fb-rib-info-model]

1.1. Definition of I2RS Filter Based RIB

Filter-based routing is a technique used to make packet forwarding decisions based on a filter that is matched to the incoming packets and the specified action. It should be noted that that this is distinct from the static routes in the RIB [I-D.ietf-i2rs-rib-info-model] where the routing is destination ddress based.

A Filter-Based RIB (Routing Information Base) is contained in a routing instance (defined in [I-D.ietf-i2rs-rib-info-model]). It contains a list of filters (match-action conditions) and a list of interfaces the filter-based forwarding operates on, and default RIB(s).

A Filter Based RIB uses packet forwarding policy. If packet reception is considered an event, then the I2RS Filter-based RIB uses a minimalistic Event-matchCondition-Action policy with the following characteristics:

event = packet/frame received,

match condition - match on field in frame/packet or circumstances relating to packet reception (e.g. time received),

action - modify packet and forward/drop packet.

A Filter-based RIB entry specifies match filters for the fields in a packet (which may include layer 1 to layer 3 header fields, transport or application fields) or size of the packet or interface received on. The matches are contained in an ordered list of filters which contain pairs of match condition-action (aka event-condition-action).

If all matches fail, default action is to forward the packet using Destination Based forward from the default RIB(s). The default RIBs can be:

- created by the I2RS Routing Informational Base (RIB) manager using the yang model described in: in [I-D.ietf-i2rs-rib-info-model], or
- configured RIB created using static routes or [I-D.ietf-netmod-routing-cfg].
- or static RIB created via static route yang model

Actions in the condition-action pair may impact forwarding or set something in the packet that will impact forwarding. Policy actions

are typically applied before applying QoS constraints since policy actions may override QoS constraint.

The Filter-Based RIB resides in ephemeral state as does the I2RS RIB and I2RS topology models.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

3. Definitions and Acronyms

CLI

Command Line Interface

FB-RIB

Filter-Based Routing Information Base

FB-Route

The policy rules in the filter-based RIB are prescriptive of the Event-Condition-Action form which is often represented by if Condition then action".

Policy Group

Policy Groups are groups of policy rules. The groups of policy in the basic network policy [I-D.hares-i2rs-pkt-eca-data-model] allow grouping of policy by name. This structure allow easier management of customer-based or provider based filters, but does not change the policy-rules list.

RIB IM

RIB Informational Model (RIB IM) [I-D.ietf-i2rs-rib-info-model]

Routing instance

A routing instance, in the context of the FB-FIB is a collection of RIBs, interfaces, and routing parameters. A routing instance

creates a logical slice of the router and allows different logical slices; across a set of routers; to communicate with each other.

4. High level Yang structure for the FB-RIB

There are three levels in the Filter-Based RIB (FB-RIB) structure:

- a global FB-RIB structures,
- the common structure of the FB-RIB, and
- the groupings that make up the FB-RIB

All structures have two types: configuration/ephemeral state and operational state.

This yang model describes three types of FB-RIBS: configuration, I2RS, and BGP Flow Specification. The configuration FB-RIB yang module is config state ("config true" and "ephemeral false") and survives a reboot. The I2RS FB-RB yang model is reboot ephemeral ("config true" and "ephemeral true;"). The BGP Flow Specification Filter-Based RIB stores policy which is received by the BGP peers. The BGP configuration contains a flow-specification as part of its configuration, and the ability to export this flow specification to other BGP peers. The BGP local configuration with the flow specification is consider ("config true"), and it is possible to be ephemeral ("ephemeral true") or local configuration ("ephemeral false"). The BGP flow specifications received from peers are derived state, just like other BGP dervised state.

Configuration RIBS

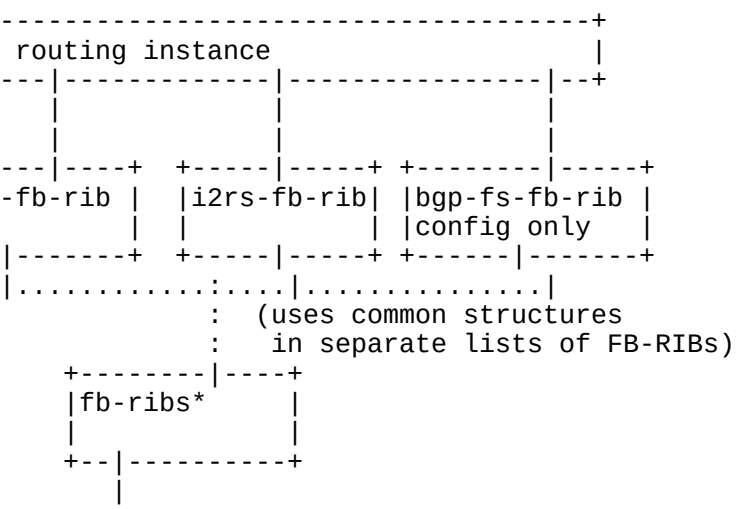


Figure 3: Routing instance with three types of Filter-FIB lists

The following section provides the high level yang structure diagrams for the following levels of structures for both config/ephemeral state and operationa.

- ietf-fb-rib - contains filter-based RIBS for config, I2RS FB-RIB, and BGP Flow Specification.
- fb-rib - that contains the structures for the filter-based grouping
- fb-rib-types - that contains the structures for groupings within the filter-based RIBS

These structures are contained within the yang section in this draft.

The packet-reception ECA policy yang module is contained in the draft [I-D.hares-i2rs-pkt-eca-data-model].

For those who desire more information regarding the logic behind the I2RS Filter-Based RIB, please see the Informational Model at: [I-D.kini-i2rs-fb-rib-info-model].

4.1. Top Level Yang Structure for ietf-fb-rib

The Top-level Yang structure for a global FB-RIB types (similar to acl) is not defined for filter-based RIBS. The I2RS Filter-Based RIB should be defined under this structure under a routing instance. The three things under this RIB would be: configured Filter-Based RIB (aka Policy routing), I2RS reboot Ephemeral Filter-Based RIB, and BGP Flow Specification's Filter-Based RIB. All of these RIBs have similar actions.

There are two types top-level structures for ietf-fb-ribs: config and operational state.

The Top-level Yang structure for a global configuration of Filter-Based RIBs are:

```
Augments rt:logical-network-elements:\
        :logical-network-element:network-instances: \
            network-instance


ietf-fb-rib module
  +--rw ietf-fb-rib
     +--rw default-instance-name string
     +--rw default-router-id rt:router-id
     +--rw config-fb-ribs
            if-feature "config-filter-based-RIB";
        uses fb-ribs;
     +--rw i2rs-fb-ribs
                 if-feature "I2RS-filter-based-RIB";
                 uses fb-rib-t:fb-ribs;
     +--rw bgp-fs-fb-ribs
                if-feature "BGP-FS-filter-based-RIB";
                 uses fb-rib-t:fb-ribs;
```

Figure 5: configuration state

The Top-level Yang structure for a global operational state of Filter-Based RIBs are:

```
   Augments rt:logical-network-elements:\
           :logical-network-element:network-instances: \
               network-instance

   ietf-fb-rib module
     +--rw ietf-fb-rib-opstate
       +--rw default-instance-name string
       +--rw default-router-id rt:router-id
           +--rw config-fb-rib-opstate
                     if-feature "config-filter-based-RIB";
                     uses fb-rib-t:fb-ribs-oper-status;
           +--rw i2rs-fb-rib-opstate {
                     if-feature "I2RS-filter-based-RIB";
                     uses fb-rib-t:fb-ribs-oper-status;
           +--rw bgp-fs-fb-rib-opstate
                     if-feature "BGP-FS-filter-based-RIB";
                     uses fb-rib-t:fb-ribs-oper-status;
```

Figure 5: operational state

4.2. Filter-Based RIB structures

The Top-level yang structures at the Filter-Based RIB level have two types: configuration and operational state.

The Top-level Yang structure for the FB-RIB types is:

```
    module: fb-rib-types:
    +--rw fb-ribs
       +--rw fb-rib* [rib-name]
       |  +--rw rib-name string
       |  |  rw fb-type identityref / ephemeral or not
       |  +--rw rib-afi rt:address-family
       |  +--rw fb-rib-intf* [name]
       |  |  +--rw name string
       |  |  +--rw intf if:interface
       |  +--rw default-rib
       |  |  +--rw rt-rib string
       |  |  +--rw config-rib string;  // config rib name
       |  |  +--rw i2rs-rib:routing-instance:name
       |  |  +--rw i2rs-rib string;   //ephemeral rib name
       |  |  +--rw bgp-instance-name string
       |  |  +--rw bgp-rib  string    //session ephemeral
       |  +--rw fb-rib-refs
       |  |  +--rw fb-rib-update-ref uint32 /count of writes
       |  +--rw instance-using*
       |  |   device:networking-instance:networking-instance-name
       |  +--use pkt-eca:pkt-eca-policy-set

   rt-rib - refer to static rib.
```

Figure 6: FB RIB Type Structure

```
   HIgh Level Yang

   +--rw fb-ribs-oper-status
      +--rw fb-rib-oper-status* [fb-rib-name]
            uses pkt-eca:pkt-eca-opstate
```

5. yang models

5.1. Filter-Based RIB types

```
 <CODE BEGINS> file "ietf-fb-rib-types@2016-02-09.yang"
  module ietf-fb-rib-types {

   yang-version "1";

   // namespace
     namespace "urn:ietf:params:xml:ns:yang:ietf-fb-rib-types";
     prefix "fb-rib-t";
         import ietf-interfaces {prefix "if";}
         import ietf-routing {prefix "rt";}
         import ietf-pkt-eca-policy {prefix "pkt-eca";}
```

```
   // meta
   organization
     "IETF";

   contact
      "email: shares@ndzh.com;
           email: sriganesh.kini@ericsson.com
       email: cengiz@packetdesign.com
       email: ivandean@gmal.org
       email: linda.dunbar@huawei.com;
       email: russ@riw.com;
          ";

   description
     "This module describes a YANG model for the I2RS
     Filter-based RIB Types.  These types
     specify types for the Filter-Based RIB.

         Copyright (c) 2015 IETF Trust and the persons identified as
     the document authors.  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
     (http://trustee.ietf.org/license-info).";


   revision "2016-02-09" {
       description
         "Filter-Based RIB protocol ";
         reference "draft-hares-i2rs-fb-rib-data-model-01";
     }

     typedef fb-rib-type-def {
            type identityref {
              base "fb-rib-type";
                  }
                description
                "This type is used to refer to
                 source of Filter-Based RIB:
                 configuration, I2RS, Flow-Spec.";
      }

         identity fb-rib-type {
                description
                "This type is used to refer to
```

```
                source of Filter-Based RIB:
                configuration, I2RS, Flow-Spec.";
        }

        identity fb-rib-config-type {
                base fb-rib-type;
             description
             "config Filter-Based RIB";
         }

        identity fb-rib-i2rs-ephemeral-type {
                base fb-rib-type;
             description
             "I2RS Reboot ephemeral Filter-Based RIB";
         }

        identity fb-rib-BGP-FS-type {
                base fb-rib-type;
             description
             "BGP Flow Specification Filter-Based RIB";
         }

     typedef fb-rib-policy-type-def {
            type identityref {
              base "fb-rib-policy-type";
                  }
                 description
                 "This type is used to refer to FB-RIB type";
      }

        identity fb-rib-policy-type {
             description
             "Types of filter-based policies
                  acl and eca";
         }

           identity fb-rib-acl {
                  base fb-rib-policy-type;
              description
           "filter based policy based on access-lists";
             }

           identity fb-bnp-eca-rules {
                  base fb-rib-policy-type;
              description
           "filter based policy based on qos forwarding rules";
             }
```

```
        typedef fb-rules-status  {
           type identityref {
             base "fb-rule-opstat";
                 }
               description
               "This type is used to refer to FB-RIB type";
         }

          identity fb-rule-opstat {
                description
                "operational statues for filter rules
                 inactive and active";
                 }

          identity fb-rule-inactive {
                base fb-rule-opstat;
                description
                "policy rule is inactive";
           }

          identity fb-rule-active {
                base fb-rule-opstat;
                description
                "policy rule is active";
           }

          grouping fb-rib-rule-order-status {
          leaf statement-order {
                  type uint16;
                  description "order identifier";
           }
           leaf statement-oper_status {
                   type fb-rules-status;
                   description "status of rule";
                }
                description "filter-rib
                  policy rule order and status";
          }

      grouping fb-rib-group-order-status {
           leaf group-refcnt {
             type uint16;
             description "refcnt for this group";
                 }
            leaf group-installed {
             type uint32;
             description "number of rules installed";
                 }
```

```
                leaf group-matches {
                 type uint64;
                 description "number of matches by all
                  rules in group";
                }
                description "fb-rib group list order
                   and status info.";
          }

        grouping fb-rib-updates {
           leaf fb-rib-update-ref {
                 type uint64;
             description
                 "number of updates to this FB RIB
                  since last reboot";
           }
           description "FB-RIB update info";
         }

        grouping default-fb-rib {
           // configuration instance for default RIB
        leaf config-instance {
                type string;
                description "instance name - string until
                   netmod fixes mount issues";
                }
                leaf config-rib {
                  type string;
                  description "name of config default RIB";
                }
                //I2RS default instance for default RIB
            leaf i2rs-instance-name {
                 type string;
                 description "I2RS instance name";
                }
                leaf i2rs-rib-name {
                        type string;
                description "name of default I2RS RIB";
                }
                leaf bgp-instance-name {
                   type string;
                   description "name of bgp instance";
                }

                leaf bgp-fs-rib-name {
                    type string;
                        description "name of BGP
                         flow specification default RIB";
```

```
                }
            description "default RIB for forwarding
                   if the policy match";
    }

    grouping fb-ribs {
            list fb-rib {
                    key fb-rib-name;
                    leaf fb-rib-name {
                        type string;
                                mandatory true;
                        description "RIB name";
            }
                    uses rt:address-family;
                    leaf fb-type {
                           type fb-rib-type-def;
                           description "type of RIB
                               list: config, I2RS rebooot
                               ephemeral, BGP Flow Specification
                               ephemeral. ";
                    }
                list fb-rib-intf {
                          key "name";
                          leaf name {
                                type if:interface-ref;
                            description
                             "A reference to the name of a
                                 configured network layer
                              interface.";
                             }
                          description "This represents
                            the list of interfaces
                            associated with this routing instance.
                            The interface list helps constrain the
                            boundaries of packet forwarding.
                            Packets coming on these interfaces are
                            directly associated with the given routing
                            instance. The interface list contains a
                            list of identifiers, with each identifier
                            uniquely identifying an interface.";
                }
                        uses default-fb-rib;  // defaults ribs
                        uses fb-rib-updates;  // write refs to this RIB
                list instance-using {
                          key instance-name;
                          leaf instance-name {
                            type string;
                            description
```

```
                                  " name of instance using this fb-rib
                                   rt:routing-instance";
                               }
                              description "instances using
                               this fb-rib";
                            }
                        // ordered rule list + group list
                        uses pkt-eca:pkt-eca-policy-set;

                    description  "Configuration of
                       an filter-based rib list";
                }
               description "fb-rib group";
        }

            grouping fb-ribs-oper-status {
                list fb-rib-oper-status {
                  key fb-rib-name;
                      leaf fb-rib-name {
                              type string;
                              description "rib name";
                      }
              uses pkt-eca:pkt-eca-opstate;
                      description  "Configuration of
                         an filter-based rib list";
                }
               description "list of FB-FIB operational
                 status";
        }


  }

 <CODE ENDS>
```

5.2. FB-RIB

```
 <CODE BEGINS> file "ietf-fb-rib@2016-02-09.yang"
module ietf-fb-rib {
  yang-version "1";

  // namespace
  namespace "urn:ietf:params:xml:ns:yang:ietf-fb-rib";
  // replace with iana namespace when assigned
    prefix "fb-rib";


  // import some basic inet types
```

```
  import ietf-yang-types {prefix "yang";}
  import ietf-fb-rib-types { prefix "fb-rib-t";}

  // meta
  organization
    "IETF";

  contact
     "email: sriganesh.kini@ericsson.com
          email: cengiz@packetdesign.com
      email: anoop@ieee.duke.edu
      email: ivandean@gmail.org
      email: shares@ndzh.com;
      email: linda.dunbar@huawei.com;
      email: russ@riw.com;
         ";

  description
    "This Top level module describes a YANG model for the I2RS
       Filter-based RIB which is an global protocol independent FB RIB module."
;

     revision "2016-02-09" {
       description "initial revision";
       reference "draft-hares-i2rs-fb-rib-data-model-01";
     }

        feature config-filter-based-RIB {
    description
      "This feature means that a node support
       config filter-based rib.";
    }
        feature I2RS-filter-based-RIB {
    description
      "This feature means that a node support
       I2RS filter-based rib.";
    }
        feature BGP-FS-filter-based-RIB {
    description
      "This feature means that a node support
      BGP FS filter-based rib.";
    }


         container ietf-fb-rib {
           presence "top-level structure for
            configuration";
       leaf default-instance-name {
            type string;
```

```
             mandatory true;
        description
          "A routing instance is identified by its name,
           INSTANCE_name.  This MUST be unique across all routing
           instances in a given network device.";
         }
             leaf default-router-id {
                    type yang:dotted-quad;
                    description "Default router id";
                }
                container config-fb-rib {
                 if-feature config-filter-based-RIB;
                  uses fb-rib-t:fb-ribs;
                 description "config filter-based RIB";
                }

                container i2rs-fb-rib {
                  if-feature I2RS-filter-based-RIB;
                  uses fb-rib-t:fb-ribs;
                 description "bgp-fs filter-based RIB";
                }
                container bgp-fs-fb-rib {
                  if-feature BGP-FS-filter-based-RIB;
                  uses fb-rib-t:fb-ribs;
                 description "bgp fs filter-based RIB";
                }
           description "fb-rib augments routing instance";
         }

        container ietf-fb-rib-opstate {
           presence "top-level structure for
            op-state";
           config "false";
       leaf default-instance-name {
             type string;
                 mandatory true;
        description
          "A routing instance is identified by its name,
           INSTANCE_name.  This MUST be unique across all routing
           instances in a given network device.";
         }
             leaf default-router-id {
                    type yang:dotted-quad;
                    description "Default router id";
                }
                container config-fb-rib-opstate {
                  if-feature config-filter-based-RIB;
                  uses fb-rib-t:fb-ribs-oper-status;
```

```
                 description "config filter-based RIB";
                }
                container i2rs-fb-rib-opstate {
                  if-feature I2RS-filter-based-RIB;
                  uses fb-rib-t:fb-ribs-oper-status;
                 description "bgp-fs filter-based RIB";
                }
                container bgp-fs-fb-rib-opstate {
                  if-feature BGP-FS-filter-based-RIB;
                  uses fb-rib-t:fb-ribs-oper-status;
                 description "bgp fs filter-based RIB";
                }
           description "fb-rib augments routing instance";
         }
}


<CODE ENDS>
```

6. IANA Considerations

TBD

7. Security Considerations

A I2RS RIB is ephemeral data store that will dyanamically change traffic paths set by the routing configuration. An I2RS FB-RIB provides dynamic Event-Condition-Action policy that will further change the operation of forwarding by allow dyanmic policy and ephemeral RIBs to alter the traffic paths set by routing configuration. Care must be taken in deployments to use the appropriate security and operational control to make use of the tools the I2RS RIB and I2RS FB-RIB provide.

8. References

8.1. Normative References:

[I-D.hares-i2rs-pkt-eca-data-model]
Hares, S., Wu, Q., and R. White, "Filter-Based Packet Forwarding ECA Policy", draft-hares-i2rs-pkt-eca-data-model-02 (work in progress), February 2016.

[I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-15 (work in progress), April 2016.

[I-D.ietf-i2rs-rib-data-model]
Wang, L., Ananthakrishnan, H., Chen, M., amit.dass@ericsson.com, a., Kini, S., and N. Bahadur, "A YANG Data Model for Routing Information Base (RIB)", draft-ietf-i2rs-rib-data-model-05 (work in progress), March 2016.

[I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-08 (work in progress), October 2015.

[I-D.ietf-netmod-acl-model]
Bogdanovic, D., Koushik, K., Huang, L., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-07 (work in progress), March 2016.

[I-D.ietf-netmod-routing-cfg]
Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-21 (work in progress), March 2016.

[I-D.kini-i2rs-fb-rib-info-model]
Kini, S., Hares, S., Dunbar, L., Ghanwani, A., Krishnan, R., Bogdanovic, D., and R. White, "Filter-Based RIB Information Model", draft-kini-i2rs-fb-rib-info-model-03 (work in progress), February 2016.

8.2. Informative References

[I-D.acee-rtgwg-yang-rib-extend]
Lindem, A. and Y. Qu, "RIB YANG Data Model", draft-acee-rtgwg-yang-rib-extend-01 (work in progress), March 2016.

[I-D.ietf-i2rs-usecase-reqs-summary]
Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", draft-ietf-i2rs-usecase-reqs-summary-02 (work in progress), March 2016.

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

Authors' Addresses

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Sriganesh Kini
Ericsson

Email: sriganesh.kini@ericsson.com

Linda Dunbar
Huawei
USA

Email: linda.dunbar@huawei.com

Ram Krishnan
Dell

Email: Ramkri123@gmail.com

Dean Bogdanovic
Juniper Networks
Westford, MA

Email: ivandean@gmail.org

Russ White
Linkedin

Email: russ@riw.us

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: December 18, 2016

S. Kini
Ericsson
S. Hares
L. Dunbar
Huawei
A. Ghanwani
R. Krishnan
Dell
D. Bogdanovic
Juniper Networks
R. White
Linkedin

June 16, 2016

Filter-Based RIB Information Model
draft-ietf-i2rs-fb-rib-info-model-00

Abstract

This document defines an information model associated with the I2RS ephemeral state for filter-based routing of IP packets via a Filter-based Routing Information Base (FB-RIB). FB-RIBs (ephemeral and non-ephemeral) are associated with specific interfaces interfaces on a routing device, and process packets received on these interfaces according a filtering policy. A filtering policy is a a minimalistic event-match_condition-action (ECA) policy with only one event - the reception of a frame/packet of data on an interface. The match conditions in the filter policy are n-tuple matches based on the content of the frame/packet or the time of its arrival. Filter-based policy allows actions which modifying the frame/packet, forward the frame or packet, or drop the frame/packet. Filter-Based Policy in FB-RIBs engages before any destination based routing so the FB-RIBs provide a destination-based default RIB that will be used if none of the filters are matched.

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 December 18, 2016.

Table of Contents

1. Introduction

The Interface to the Routing System (I2RS) [I-D.ietf-i2rs-architecture] architecture provides dynamic read and write access to the information and state within the routing elements. The I2RS client interacts with the I2RS agent in one or more network routing systems.

This document provides an information module for Filter Based Routing Information Base (FB-RIB) and describes the I2RS interaction with routing filters within a routing element.

1.1. Definition of I2RS Filter Based RIB

Filter-based routing is a technique used to make packet forwarding decisions based policy-based filters that are matched to the incoming packets. These filter policies are a minimalistic "event-Match Condition-Action" policy with one event - the reception of a frame or packet on an associated interface. The ECA policies have:

- o event = reception of frame/packet on associated interface,
- o match condition = policy filters which match portions of frame/ packet,
- o actions - to modify frame/packet, and forward or drop frame/packet

Filter-Based forwarding may match on the condition of any portion of the frame/packet. Figure 1 shows some of the filters that can be applied to the reception of packets. The filter for individual fields can be combined with an "AND" or and "OR", but the default combination is that of an "AND".

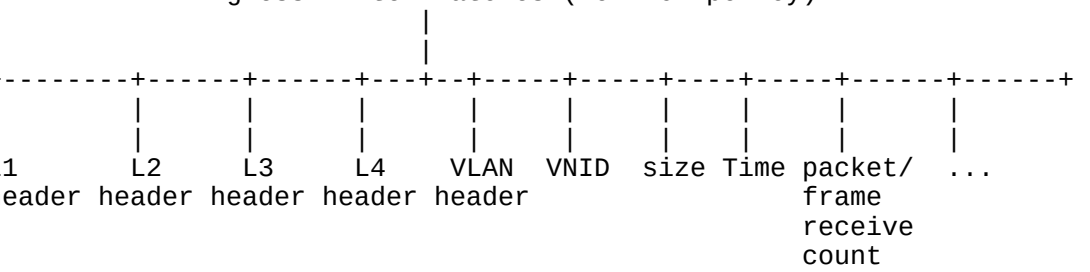


Figure 1: Possible matching conditions for basic network filters

A Filter-Based Routing Information Base (FB-RIB)) is a set of packet-reception ECA policy rules which are:

- ordered, and
- apply to specific interfaces

If all matches fail, default action is to forward the packet using a specific RIB from:

- routing RIBs as described in [I-D.ietf-netmod-routing-cfg]"
- ephemeral I2RS RIBs as described in [I-D.ietf-i2rs-rib-info-model].

1.2. I2RS Use Cases Suported by Filter-Based RIB

The I2RS use cases which benefit from Filter-Based Routing are:

- Protocol independent Use cases and large flow use cases described in [I-D.hares-i2rs-usecase-reqs-summary].
- the use cases of steering traffic to their designated service functions that are different than the packet’s destinations, and
- large flow use cases described in [I-D.hares-i2rs-usecase-reqs-summary]

1.3. Definitions and Acronyms

CLI

Command Line Interface

FB-RIB

Filter-Based Routing Information Base

FB-Filter

One filter in the filter-based RIB. The filters are Event-Condition-Action filters often represented by the form "if Condition then action".

Policy Group

Policy Groups are groups of policy rules. The groups of policy in the basic network policy [I-D.hares-i2rs-pkt-eca-data-model] allow grouping of policy by name. This name allow easier management of customer-based or provider based filters.

RIB IM

RIB Informational Model (RIB IM) is an information model which describes a Routing Information Base

Routing instance

A routing instance is a core data model within [I-D.ietf-netmod-routing-cfg]. An instance creates a logical slice of the router and allows different logical slices to communicate to communicate with each other.

2. I2RS Filter-Based RIB place among Filter-Based RIBS

The following three types of Filter-Based RIBs exist:

- Policy Based Routing - configured by packet ECA policy,
- I2RS Filter-Based RIBs - proposed by this document,
- BGP Flow Specification [RFC5575].

This section examines issues regarding ephemeral versus config state, the need for order within policies, and the current yang support for packet-reception ECA policy.

2.1. Ephemeral State versus Configuration State

Filter-Based RIBs with packet-reception ECA policy exist in three types of state: configuration state, ephemeral reboot state (I2RS), and BGP Session state.

Policy Routing configures the packet-reception ECA policy in configuration state, and runs this policy to specific interfaces. This configuration state may be configured by NETCONF/RESTCONF in yang modules or proprietary configuation via CLI. This specification examines configuration state specified in yang modules and extended by proprietary additions to yang modules. Yang modules which specify the normal routing RIBs include:

[I-D.ietf-netmod-routing-cfg]

statics routes (TBD)

Configuration state set via secure protocols (NETCONF [RFC6241] or RESTCONF [I-D.ietf-netconf-restconf]) and survives the reboot of the system. draft-hares-rtgwg-fb-rib refers to Filter-Based RIB described above which contains an ordered list of packet

I2RS ephemeral state [I-D.ietf-i2rs-ephemeral-state] is state which does not persist across a reboot (software or hardware). I2RS ephemeral state can be indicated a portion of a sub-tree, a sub-tree, tree within a yang modules, or a full module. I2RS Ephemeral state may reference configuration state or protocol state (E.g. MPLS LSP or BGP peer state).

The I2RS Filter-Based RIB is defined as a ephemeral module with possible links to the following:

default RIBs either in the I2RS RIB module [I-D.ietf-i2rs-rib-data-model] or configured RIB [I-D.ietf-netmod-routing-cfg],

interfaces that I2RS Filter-Based RIB is running on.

BGP Flow Specification [RFC5575] passes packet-reception ECA Policy in BGP UPDATE packets (NLRI and BGP Extended Communities). The BGP Flow Specification packet-reception ECA policy is bgp peer session ephemeral state which disappears when the BGP peer closes the BGP Session. This bgp session ephemeral state can refer to configuration state for interfaces configured, and for default RIBs. [RFC5575] does not consider I2RS configuration state.

Precedence between these three types of state is defined as most dynamic to least dyanmic, or

1. BGP Session packet-reception Filter Policy,

2. I2RS Filter-Based RIB Policy,

3. Configuration Filter-RIB Policy (aka Policy RIB).

Precedence impacts when two types of state try to operate on the exact same filter match in the policy. However, Filter-Based RIBS operate first on "order" and priority within the order, and then on the level of ephemeral state.

2.2. Need for Order

Filter-Based RIBs use packet-reception ECA policy instead of destination based forwarding to determine where to forward/send a packet. A packet which matches multiple filters in a Filter-Based RIB will be forwarded based on the first filter matched. Due to first-match processing, the order of filters matters in the process. All Filter-Based RIBs (configuration, I2RS, and BGP Flow Specification) forwarded based on the first match filter.

Filter-Based Policy is inserted in the RIB by order number. If order number is tied, then precedence is based on the type of filter, longest prefix match (MAC or IP address (IPv4/IPv6), lowest value, or longest string). It is expected that order will contain a large enough number space to differentiate most policies.

Note: [RFC5575] does not support order, but current work is beginning draft-hares-idr-flow-spec-combo-00.txt to support order in BGP flow specification.

```
flow-rule-cmp (a,b)
  {
      comp1 = next_component(a);     /component is the type of filter
      comp2 = next_component(b);
   while (comp1 || comp2) {
    // component_type returns infinity on end of list
     if (component_type(comp1) < component_type(comp2)) {
      return A_HAS_PRECEDENCE;
     }

     if (component_type(comp1) > component_type(comp2)) {
     return B_HAS_PRECEDENCE;
     }
     // IP values)
     if (component_type(comp1) == IP_DESTINATION || IP_SOURCE) {
      common = MIN(prefix_length(comp1),prefix_length(comp2));
          cmp = prefix_compare (comp1,comp2,common);
          // not equal, lowest value has precedence
          // equal, longest match has precedence;
     } else if (component_type (comp1) == MAC_DESTINATION ||
            MAC_SOURCE) {
              common = MIN(MAC_address_length(comp1),
                           MAC_address_length(comp2));
              cmp = MAC_Address_compare(comp1,comp2,common);
              //not equal, lowest value has precedence
              //equal, longest match has precedence
         } else {
      common = MIN(component_length(comp1),
                           component_length(comp2));
          cmp = memcmp(data(comp1), data(comp2), common);
              //not equal, lowest value has precedence
              //equal, longest string has precedence
     }
    }
   }
```

Figure 2 - precedence

2.3. ECA Policy Supported

The filter based-RIB uses event-condition-action policy (ECA) rules. The following policies are used in this version of the yang module:

- Access lists (ACLs) [I-D.ietf-netmod-acl-model]
- Packet-reception ECA policy [I-D.hares-i2rs-pkt-eca-data-model]

Proprietary filters may augment these IETF defined ECA rules. The IETF filters support basic filtering plus QOS and load balancing. Below is an example set of match conditions on ingreessI2RS that the basic I2RS FB-RIB can support.

2.4. Relationship between Filter-RIBs and RIBS

Filter-based RIBS (FB-RIBs) provide packet-reception event-match condition-action policy, but if the filters do not provide match the Filter-Based RIBs provide default RIBs for destination based forwarding (IP or MAC). The following are restrictionsThe for the default RIBs:

- The configuration FB-RIBs can only refer to another configuration RIB.
- The I2RS FB-RIBs can refer to a configuration RIB, an I2RS reboot ephemeral RIB, or a BGP Session ephemeral RIB. The BGP peer session may be dropped while a I2RS FB-RIB is in session. If so, all defaults pointing to the BGP RIB must be removed. The I2RS RIB may be removed, and if so all defaults pointing to that route must be removed. The default order of precedence for the default RIB is the BGP-Peer default RIB, the I2RS default FB-RIB, and the configuration default RIB.
- The BGP session FB-RIBs can refer to a configuration RIBs, a I2RS Ephemeral RIB, and a BGP RIB. Just as with the I2RS FB-RIB, the precedence if multiple default RIBs exist are: BGP Peer default RIB, then I2RS default RIB, followed by configuration default RIB.
- The I2RS Ephemeral RIB module is described in [I-D.ietf-i2rs-rib-info-model] and [I-D.ietf-i2rs-rib-data-model]. The I2RS RIB contains a collection of RIBs with the following information per instance:
  - The set of interfaces indicates which interfaces are associated with this routing instance.

    * The RIBs specify how incoming traffic is to be forwarded based on destination (E.g. RIB and FB-RIB).

    * The routing parameters control the information in the RIBs.

  o A routing instance may have both an I2RS RIB modules and I2RS FB-FIB modules associated with it. If the I2RS RIB list of interfaces does not contain the list of interfaces the FB-RIB is operating on, then the I2RS RIB must not be installed as a default RIB.

  o FB-RIB and RIB can not be used at the same time, which means:

    * If a router doesn't support filter-based routing, a router MUST use RIB and MUST not use FB-RIB. The default RIB for a FB-RIB should destination-based RIB, and this RIB may be generated by routing protocols. However, the FB-RIB forwarding must take precedence over the default RIB.

    * If a router supports filter-based routing:

      + FB-RIB is used

      + Multiple FB-RIBs may exist within a routing instance

      + An interface can be associated with at most one FB-RIB

      + The Default RIB for a FB-RIB is used if several criteria beyond destination address is not matched.

3. Filter-Based-RIB module

A Filter-Based RIB (FB-RIB)contains an ordered set of filter routes where each filter-route is a match condition followed by an action. An FB-RIB is contained in a routing-instance defined by [I-D.ietf-netmod-routing-cfg]. An FB-RIB has a list of interfaces that is a subset of the list of interfaces in the routing-instance that it is contained in. An incoming packet on an interface belonging to a FB-RIB is first handled by the FIB programmed using that FB-RIB. If no match action succeeds, then the packet is forwarded using the FIB programmed using the RIB of that routing instance.

An ordered set of filters implies that the insertion of a filter route into a FB-RIB MUST provide the ability to insert a filter route at any specific position and delete of a filter-based route at a specific position. The ability to change a filter route at a

specific position combines these two functions (delete an existing filter route rule and add a new policy rule).

Each FB-RIB is contained within a routing instance, but one routing instance (named by an INSTANCE_NAME) can contain multiple FB-RIBs. Each routing instance is associated with a set of interfaces, a router-id, and list of FB-RIBs. Each interface can be associated with at most one FB RIB.

The processing within the FB-RIB process within the routing system is expected to do the following:

- When a packet successfully matches match term/entry in a filter-route, the corresponding rule-actions are applied.
- If a packet does not match the match term/entry in the filter route, the filter route processing goes to the next term/entry in the order, and looks for a match, within the current filter or goes to the next filter in the list. This continues until either a filter route match term/entry is successfully matched, or no more filters in the list exists.
- If no match has been found within list of filters in FB-RIB list, then the packet will be forwarded using the I2RS RIB specified by the FB-RIB if one exists. If no I2RS RIB is specified, the FB-RIB will check a configured RIB. If no configured RIB exists, the packet will be discarded.

Groups within a I2RS FB-RIB allow the logical grouping of rules under a name for ease of access. For example, take two customers. Customer-A has three packet-reception ECA policies that insert rules at order 5, 10, and 20. Customer-B has three packet-reception ECA policies that insert rules at 4, 8, and 9. The use of the group names "Customer-A" and "Customer-B" allow easy addition or deletion of these rules, but do not change the ordering of these rules.

3.1. ietf-fb-rib Configuration: Top level Container

The FB-RIB configuration entries associated with each FB-RIB in a routing instance are:

default-instance-name (FB-FIB-instance-name): default Routing Instance name for all FB-RIBs

default-router-id (FB-RIB-router-id): router id associated with the FB-RIB function of the Routing instance

config-fb-rib: list of filter-based RIBs created by configuration processes, and described in draft-hares-rtgwg-fb-rib which utilizes [I-D.hares-i2rs-fb-rib-data-model] to define common filter-based RIB structures.

i2rs-fb-rib: list of I2RS reboot ephemeral filter-based RIBs. Described in this draft with data model in [I-D.hares-i2rs-fb-rib-data-model] which utilizes [I-D.hares-i2rs-fb-rib-data-model] to define common filter-based RIB structures.

bgp-fb-rib: list of BGP Session ephemeral filter-based RIBs Described in this draft, and data model in (draft-hares-idr-bgp-fb-rib-data-model). which utilizes [I-D.hares-i2rs-fb-rib-data-model] to define common filter-based RIB structures, and [I-D.hares-i2rs-pkt-eca-data-model] to the common packet-reception ECA filters.

Configuration RIBS

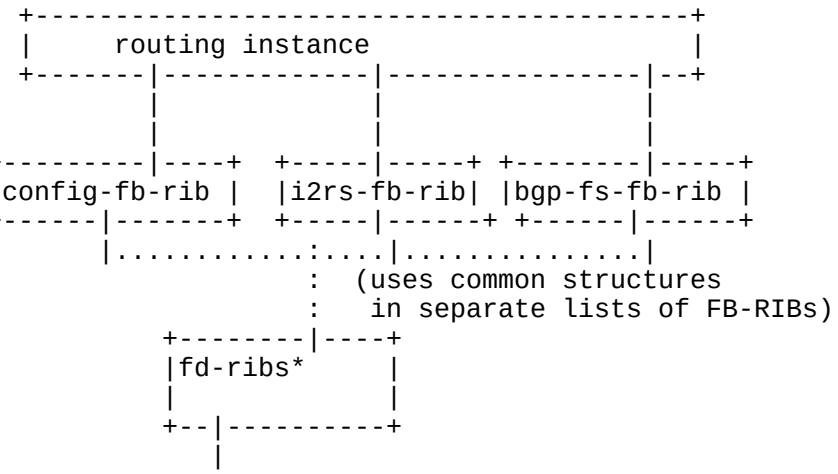


Figure 3: Routing instance with three types of Filter-FIB lists

The Top-level Yang structure for a global configuration of Filter-Based RIBs are:

```
   Augments rt:logical-network-elements:\
           :logical-network-element:network-instances: \
               network-instance


   ietf-fb-rib module
     +--rw ietf-fb-rib
        +--rw default-instance-name string
        +--rw default-router-id rt:router-id
        +--rw config-fb-ribs
               if-feature "config-filter-based-RIB";
           uses fb-ribs;
        +--rw i2rs-fb-ribs
                    if-feature "I2RS-filter-based-RIB";
                    uses fb-rib-t:fb-ribs;
        +--rw bgp-fs-fb-ribs
                   if-feature "BGP-FS-filter-based-RIB";
                    uses fb-rib-t:fb-ribs;
```

Figure 5: configuration state

3.2. ietf-fb-rib-opstate: Operational Top Level Container

The FB-RIB operational state entries associated with each FB-RIB in a routing instance are:

default-instance-name (FB-FIB-instance-name): Default Routing Instance for FB-RIBs.

default-router-id (FB-RIB-router-id): Default Router ID associated FB-RIBs.

config-fb-rib-opstate operational state for config RIB described in draft-hares-rtgwg-fb-rib which utilizes [I-D.hares-i2rs-fb-rib-data-model] to define common structures.

i2rs-fb-rib-opstate: operational state for I2RS reboot ephemeral Filter-Based RIB. Logic is described in this draft, and data model is described in [I-D.hares-i2rs-fb-rib-data-model]. Common structures are defined in [I-D.hares-i2rs-fb-rib-data-model].

bgp-fb-rib-opstate: BGP Session ephemeral Filter-Based RIB-interface with logic described in this draft, and data model in (draft-hares-bgp-fb-rib-data-model). Common structures are also defined in [I-D.hares-i2rs-fb-rib-data-model], and [I-D.hares-i2rs-pkt-eca-data-model].

Operational state

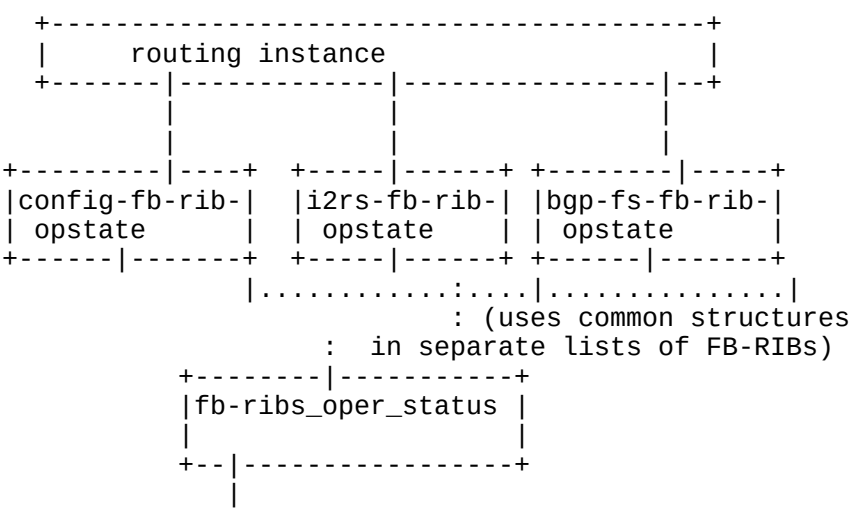


Figure 4: Routing instance with three types of Filter-FIB lists

The Top-level Yang structure for a global operational state of Filter-Based RIBs are:

```
Augments rt:logical-network-elements:\
        :logical-network-element:network-instances: \
            network-instance

ietf-fb-rib module
  +--rw ietf-fb-rib-opstate
    +--rw default-instance-name string
    +--rw default-router-id rt:router-id
        +--rw config-fb-rib-opstate
                  if-feature "config-filter-based-RIB";
                  uses fb-rib-t:fb-ribs-oper-status;
        +--rw i2rs-fb-rib-opstate {
                  if-feature "I2RS-filter-based-RIB";
                  uses fb-rib-t:fb-ribs-oper-status;
        +--rw bgp-fs-fb-rib-opstate
                  if-feature "BGP-FS-filter-based-RIB";
                  uses fb-rib-t:fb-ribs-oper-status;
```

Figure 5: operational state

3.3. fb-ribs: List of Filter-Based RIBs (Configuration)

Filter-Based RIB structures for configuration (fb-ribs) contain a list of fb-rib structures with the following high-level structure:

fb-rib-name: Name of fb-Rib (key),

address-family: AFI for Address Family,

fb-type: type of FB-RIB (config, I2RS reboot ephemeral, or BGP Flow Specification session ephemeral).

Interface_list(FB-RIB-interface): A list of interfaces that all of the FB-RIB RIB operates over. This list must be a subset of the interface_list associated with the routing instance.

default-RIBS: structure with default RIBS in configuration space, I2RS RIB space, or BGP VPN space.

instance-using: list of instances using this FB-RIB (normally one).

fb-rb-updates: Tracking Write-References to this RIB.

uses pkt-eca:pkt-eca-policy-set: pkt ECA Policy described in [I-D.hares-i2rs-pkt-eca-data-model]

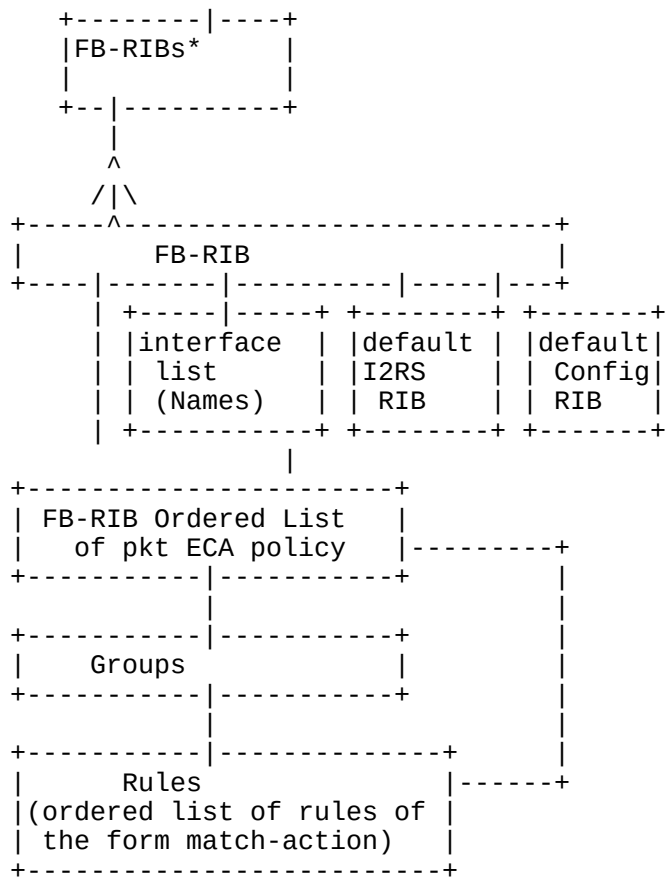


Figure 5: fb-rib (configuration FB-RIB)

The Top-level Yang structure for the FB-RIB types is:

```
    module: fb-rib-types:
    +--rw fb-ribs
       +--rw fb-rib* [rib-name]
       |  +--rw rib-name string
       |  |  rw fb-type identityref / ephemeral or not
       |  +--rw rib-afi rt:address-family
       |  +--rw fb-rib-intf* [name]
       |  |  +--rw name string
       |  |  +--rw intf if:interface
       |  +--rw default-rib
       |  |  +--rw rt-rib rt:routing:routing-instance:name
       |  |  +--rw config-rib string;  // config rib name
       |  |  +--rw i2rs-rib:routing-instance:name
       |  |  +--rw i2rs-rib string;   //ephemeral rib name
       |  |  +--rw bgp-instance-name string
       |  |  +--rw bgp-rib  string    //session ephemeral
       |  +--rw fb-rib-refs
       |  |  +--rw fb-rib-update-ref uint32 /count of writes
       |  +--rw instance-using*
       |  |   device:networking-instance:networking-instance-name
    |  +--use pkt-eca:pkt-eca-policy-set
```

Figure 6: FB RIB Type Structure

3.4. fb-rib-oper-status - list of filter-ribs with operational status

The Filter-Based RIB structures for operational state have the following entries:

fb-rib-name: Name of fb-Rib (key)

uses pkt-eca:pkt-eca-opstate: pkt ECA policy operational state described in [I-D.hares-i2rs-pkt-eca-data-model]

HIgh Level Yang

```
+--rw fb-ribs-oper-status
   +--rw fb-rib-oper-status* [fb-rib-name]
         uses pkt-eca:pkt-eca-opstate
```

3.5. Packet-reception Event-Condition-Action Policy

The three levels of policy are expressed as:

```
Config Policy definitions
=========================================
Policy level: pkt-eca-policy-set
group level:  pkt-eca-policy-set:groups
rule level:   bnp-eca-policy-set:rules

Operational State for Policy
=========================================
Policy level: pkt-eca-policy-opstate
group level:  pkt-eca-policy-opstate:groups-status
rule level:   bnp-eca-policy-opstate:rules_opstate*
              bnp-eca-policy-opstate:rules_opstats*
```

figure

High level Yang structure for Configuration and operational status are shown in figure x below.

```
Packet Reception ECA policy
module ietf-pkt-eca-policy
  +--rw pkt-eca-policy-cfg
  |  +--rw pkt-eca-policy-set
  |     +--rw groups* [group-name]
  |     |  +--rw vrf-name string
  |     |  +--rw address-family
  |     |  +--rw group-rule-list* [rule-name]
  |     |  |  +--rw rule-name
  |     |  |  +--rw rule-order-id
  |     +--rw rules [order-id rule-name]
  |        +--rw eca-matches
  |        | ...
  |        +--rw eca-qos-actions
  |        | ...
  |        +--rw eca-fwd-actions
  |        | ...
  +--rw pkt-eca-policy-opstate
     +--rw pkt-eca-opstate
        +--rw groups* [group-name]
        |  +--rw rules-installed;
        |  +--rw rules_status* [rule-name]
        +--rw rule-group-link* [rule-name]
        |  +--rw group-name
        +--rw rules_opstate* [rule-order rule-name]
        |  +--rw status
        |  +--rw rule-inactive-reason
        |  +--rw rule-install-reason
        |  +--rw rule-installer
        |  +--rw refcnt
        +--rw rules_op-stats* [rule-order rule-name]
           +--rw pkts-matched
           +--rw pkts-modified
           +--rw pkts-forwarded
```

Figure 4 - High-Level Yang structure.

4. IANA Considerations

This informational draft does not specify any IANA requests.

5. Security Considerations

A I2RS defines an ephemeral data store that will dyanamically change traffic paths set by the routing configuration. An I2RS FB-RIB provides dynamic Event-Condition-Action policy that will further change the operation of forwarding by allow dyanmic policy and ephemeral RIBs to alter the traffic paths set by routing

configuration. Care must be taken in deployments to use the appropriate security and operational control to make use of the tools the I2RS RIB and I2RS FB-RIB provide.

6. References

6.1. Normative References:

[I-D.hares-i2rs-fb-rib-data-model]
Hares, S., Kini, S., Dunbar, L., Krishnan, R., Bogdanovic, D., and R. White, "Filter-Based RIB Data Model", draft-hares-i2rs-fb-rib-data-model-03 (work in progress), March 2016.

[I-D.hares-i2rs-pkt-eca-data-model]
Hares, S., Wu, Q., and R. White, "Filter-Based Packet Forwarding ECA Policy", draft-hares-i2rs-pkt-eca-data-model-02 (work in progress), February 2016.

[I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-15 (work in progress), April 2016.

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-09 (work in progress), May 2016.

[I-D.ietf-i2rs-rib-data-model]
Wang, L., Ananthakrishnan, H., Chen, M., amit.dass@ericsson.com, a., Kini, S., and N. Bahadur, "A YANG Data Model for Routing Information Base (RIB)", draft-ietf-i2rs-rib-data-model-05 (work in progress), March 2016.

[I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-08 (work in progress), October 2015.

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

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

6.2. Informative References

[I-D.hares-i2rs-usecase-reqs-summary]
Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", draft-hares-i2rs-usecase-reqs-summary-02 (work in progress), May 2015.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-13 (work in progress), April 2016.

[I-D.ietf-netmod-acl-model]
Bogdanovic, D., Koushik, K., Huang, L., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-07 (work in progress), March 2016.

[I-D.ietf-netmod-routing-cfg]
Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-21 (work in progress), March 2016.

[RFC5575] Marques, P., Sheth, N., Raszuk, R., Greene, B., Mauch, J., and D. McPherson, "Dissemination of Flow Specification Rules", RFC 5575, DOI 10.17487/RFC5575, August 2009, <http://www.rfc-editor.org/info/rfc5575>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

Authors' Addresses

Sriganesh Kini
Ericsson

Email: sriganesh.kini@ericsson.com

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Linda Dunbar
Huawei
USA

Email: linda.dunbar@huawei.com

Anoop Ghanwani
Dell

Email: anoop@alumni.duke.edu

Ram Krishnan
Dell

Email: Ramkri123@gmail.com

Dean Bogdanovic
Juniper Networks
Westford, MA

Email: deanb@juniper.net

Russ White
Linkedin

Email: russ@riw.us

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2017

L. Wang
Individual
H. Ananthakrishnan
Packet Design
M. Chen
Huawei
A. Dass
S. Kini
Ericsson
N. Bahadur
Bracket Computing
July 3, 2016

A YANG Data Model for Routing Information Base (RIB)
draft-ietf-i2rs-rib-data-model-06

Abstract

This document defines a YANG data model for Routing Information Base (RIB) that aligns with the I2RS RIB information model.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

Table of Contents

1. Introduction

The Interface to the Routing System (I2RS) [I-D.ietf-i2rs-architecture] provides read and write access to the information and state within the routing process that exists inside the routing elements, this is achieved via protocol message exchange between I2RS clients and I2RS agents associated with the routing system. One of the functions of I2RS is to read and write data of Routing Information Base (RIB). [I-D.ietf-i2rs-usecase-reqs-summary] introduces a set of RIB use cases. The RIB information model is defined in [I-D.ietf-i2rs-rib-info-model].

This document defines a YANG [RFC6020][RFC6991] data model for the RIB that satisfies the RIB use cases and aligns with the RIB information model.

1.1. Definitions and Acronyms

RIB: Routing Information Base

Information Model (IM): An abstract model of a conceptual domain, independent of a specific implementation or data representation.

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node and "*" denotes a "list" and "leaf-list".
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Model Structure

The following figure shows an overview of structure tree of the ietf-i2rs-rib module. To give a whole view of the structure tree, some details of the tree are omitted. The relevant details are introduced in the subsequent sub-sections.

```
module: ietf-i2rs-rib
   +--rw routing-instance
      +--rw name              string
      +--rw interface-list* [name]
      |  +--rw name    if:interface-ref
      +--rw router-id?        yang:dotted-quad
      +--rw lookup-limit?     uint8
```

```
         +--rw rib-list* [name]
            +--rw name              string
            +--rw address-family    rib-family-def
            +--rw ip-rpf-check?     boolean
            +--rw route-list* [route-index]
            |  +--rw route-index                uint64
            |  +--rw match
            |  |  +--rw (route-type)?
            |  |     +--:(ipv4)
            |  |     |  ...
            |  |     +--:(ipv6)
            |  |     |  ...
            |  |     +--:(mpls-route)
            |  |     |  ...
            |  |     +--:(mac-route)
            |  |     |  ...
            |  |     +--:(interface-route)
            |  |        ...
            |  +--rw nexthop
            |  |  +--rw nexthop-id?           uint32
            |  |  +--rw sharing-flag?         boolean
            |  |  +--rw (nexthop-type)?
            |  |     +--:(nexthop-base)
            |  |     |  ...
            |  |     +--:(nexthop-chain) {nexthop-chain}?
            |  |     |  ...
            |  |     +--:(nexthop-replicates) {nexthop-replicates}?
            |  |     |  ...
            |  |     +--:(nexthop-protection) {nexthop-protection}?
            |  |     |  ...
            |  |     +--:(nexthop-load-balance) {nexthop-load-balance}?
            |  |        ...
            |  +--rw route-status
            |  |  ...
            |  +--rw route-attributes
            |  |  ...
            |  +--rw route-vendor-attributes
            +--rw nexthop-list* [nexthop-member-id]
               +--rw nexthop-member-id    uint32
   rpcs:
      +---x rib-add
      |  +---w input
      |  |  +---w name        string
      |  |  +---w address-family      rib-family-def
      |  |  +---w ip-rpf-check?   boolean
      |  +--ro output
      |     +--ro result uint32
      |     +--ro reason? string
```

```
      +---x rib-delete
      |  +---w input
      |  |  +---w name string
      |  +--ro output
      |     +--ro result uint32
      |     +--ro reason? string
      +---x route-add
      |  +---w input
      |  |  +---w return-failure-detail?   boolean
      |  |  +---w rib-name                 string
      |  |  +---w routes
      |  |     +---w route-list* [route-index]
      |  |        ...
      |  +--ro output
      |     +--ro success-count     uint32
      |     +--ro failed-count      uint32
      |     +--ro failure-detail
      |        +--ro failed-routes* [route-index]
      |           +--ro route-index uint32
      |           +--ro error-code? uint32
      +---x route-delete
      |  +---w input
      |  |  +---w return-failure-detail?   boolean
      |  |  +---w rib-name                 string
      |  |  +---w routes
      |  |     +---w route-list* [route-index]
      |  |        ...
      |  +--ro output
      |     +--ro success-count     uint32
      |     +--ro failed-count      uint32
      |     +--ro failure-detail
      |        +--ro failed-routes* [route-index]
      |           +--ro route-index uint32
      |           +--ro error-code? uint32
      +---x route-update
      |  +---w input
      |  |  +---w return-failure-detail?           boolean
      |  |  +---w rib-name                         string
      |  |  +---w (match-options)?
      |  |     +--:(match-route-prefix)
      |  |     |  ...
      |  |     +--:(match-route-attributes)
      |  |     |  ...
      |  |     +--:(match-route-vendor-attributes) {...}?
      |  |     |  ...
      |  |     +--:(match-nexthop)
      |  |        ...
      |  +--ro output
```

```
      |     +--ro success-count uint32
      |     +--ro failed-count uint32
      |     +--ro failure-detail
      |        +--ro failed-routes* [route-index]
      |           +--ro route-index uint32
      |           +--ro error-code? uint32
      +---x nh-add
      |  +---w input
      |  |  +---w rib-name              string
      |  |  +---w nexthop-id?           uint32
      |  |  +---w sharing-flag?         boolean
      |  |  +---w (nexthop-type)?
      |  |     +--:(nexthop-base)
      |  |     |  ...
      |  |     +--:(nexthop-chain) {nexthop-chain}?
      |  |     |  ...
      |  |     +--:(nexthop-replicates) {nexthop-replicates}?
      |  |     |  ...
      |  |     +--:(nexthop-protection) {nexthop-protection}?
      |  |     |  ...
      |  |     +--:(nexthop-load-balance) {nexthop-load-balance}?
      |  |        ...
      |  +--ro output
      |     +--ro result        uint32
      |     +--ro reason?       string
      |     +--ro nexthop-id?   uint32
      +---x nh-delete
         +---w input
         |  +---w rib-name              string
         |  +---w nexthop-id?           uint32
         |  +---w sharing-flag?         boolean
         |  +---w (nexthop-type)?
         |     +--:(nexthop-base)
         |     |  ...
         |     +--:(nexthop-chain) {nexthop-chain}?
         |     |  ...
         |     +--:(nexthop-replicates) {nexthop-replicates}?
         |     |  ...
         |     +--:(nexthop-protection) {nexthop-protection}?
         |     |  ...
         |     +--:(nexthop-load-balance) {nexthop-load-balance}?
         |        ...
         +--ro output
            +--ro result uint32
            +--ro reason? string
   notifications:
      +---n nexthop-resolution-status-change
      |  +--ro nexthop
```

```
|  |  +--ro nexthop-id?           uint32
|  |  +--ro sharing-flag?         boolean
|  |  +--ro (nexthop-type)?
|  |     +--:(nexthop-base)
|  |     |  ...
|  |     +--:(nexthop-chain) {nexthop-chain}?
|  |     |  ...
|  |     +--:(nexthop-replicates) {nexthop-replicates}?
|  |     |  ...
|  |     +--:(nexthop-protection) {nexthop-protection}?
|  |     |  ...
|  |     +--:(nexthop-load-balance) {nexthop-load-balance}?
|  |        ...
|  +--ro nexthop-state nexthop-state-def
+---n route-change
   +--ro rib-name                string
   +--ro address-family              rib-family-def
   +--ro route-index             uint64
   +--ro match
   |  +--ro (route-type)?
   |     +--:(ipv4)
   |     |  ...
   |     +--:(ipv6)
   |     |  ...
   |     +--:(mpls-route)
   |     |  ...
   |     +--:(mac-route)
   |     |  ...
   |     +--:(interface-route)
   |        ...
   +--ro route-installed-state route-installed-state-def
   +--ro route-state         route-state-def
   +--ro route-change-reason route-reason-def
```

Figure 1: Overview of I2RS RIB Module Structure

2.1. RIB Capability

RIB capability negotiation is very important because not all of the hardware will be able to support all kinds of nexthops and there might be a limitation on how many levels of lookup can be practically performed. Therefore, a RIB data model MUST specify a way for an external entity to learn about the functional capabilities of a network device.

At the same time, nexthop chains can be used to specify multiple headers over a packet, before that particular packet is forwarded. Not every network device will be able to support all kinds of nexthop

chains along with the arbitrary number of headers which are chained together. The RIB data model MUST provide a way to expose the nexthop chaining capability supported by a given network device.

This module uses the feature and if-feature statements to achieve above capability advertisement.

2.2. Routing Instance and Rib

A routing instance, in the context of the RIB information model, is a collection of RIBs, interfaces, and routing protocol parameters. A routing instance creates a logical slice of the router and can allow multiple different logical slices, across a set of routers, to communicate with each other. The routing protocol parameters control the information available in the RIBs. More detail about routing instance can be found in Section 2.2 of [I-D.ietf-i2rs-rib-info-model].

For a routing instance, there can be multiple RIBs. Therefore, this model uses "list" to express the RIBs. The structure tree is shown below:

```
+--rw routing-instance
   +--rw name              string
   +--rw interface-list* [name]
   |  +--rw name if:interface-ref
   +--rw router-id?        yang:dotted-quad
   +--rw lookup-limit?     uint8
   +--rw rib-list* [name]
      +--rw name            string
      +--rw address-family      rib-family-def
      +--rw ip-rpf-check?   boolean
      +--rw route-list* [route-index]
         ... (refer to Section 2.3)
```

Figure 2: Routing Instance Structure

2.3. Route

A route is essentially a match condition and an action following that match. The match condition specifies the kind of route (e.g., IPv4, MPLS, MAC, Interface etc.) and the set of fields to match on.

According to the definition in [I-D.ietf-i2rs-rib-info-model], a route MUST associate with the following attributes:

- ROUTE_PREFERENCE: See Section 2.3 of [I-D.ietf-i2rs-rib-info-model].

o ACTIVE: Indicates whether a route has at least one fully resolved nexthop and is therefore eligible for installation in the FIB.

o INSTALLED: Indicates whether the route got installed in the FIB.

In addition, a route can be associated with one or more optional route attributes (e.g., route-vendor-attributes).

A RIB will have a number of routes, so the routes are expressed as a list under a specific rib. Each RIB has its own route list.

```
+--rw route-list* [route-index]
   +--rw route-index               uint64
   +--rw match
   |  +--rw (route-type)?
   |     +--:(ipv4)
   |     |  +--rw ipv4
   |     |     +--rw (ip-route-match-type)?
   |     |        +--:(dest-ipv4-address)
   |     |        |  ...
   |     |        +--:(src-ipv4-address)
   |     |        |  ...
   |     |        +--:(dest-src-ipv4-address)
   |     |           ...
   |     +--:(ipv6)
   |     |  +--rw ipv6
   |     |     +--rw (ip-route-match-type)?
   |     |        +--:(dest-ipv6-address)
   |     |        |  ...
   |     |        +--:(src-ipv6-address)
   |     |        |  ...
   |     |        +--:(dest-src-ipv6-address)
   |     |           ...
   |     +--:(mpls-route)
   |     |  +--rw mpls-label             uint32
   |     +--:(mac-route)
   |     |  +--rw mac-address            uint32
   |     +--:(interface-route)
   |        +--rw interface-identifier if:interface-ref
   +--rw nexthop
   |  ...(refer to Section 2.4)
```

Figure 3: Routes Structure

2.4. Nexthop

A nexthop represents an object resulting from a route lookup. As illustrated in Section 2.4 of [I-D.ietf-i2rs-rib-info-model], to support various use cases (e.g., load balance, protection, multicast or a combination of them), the nexthop is modeled as a multi-level structure and supports recursion. The first level of the nexthop includes the following four types:

- Base: The "base" nexthop is the foundation of all other nexthop types. It includes the follow basic nexthops:
  - nexthop-id
  - IPv4 address
  - IPv6 address
  - egress-interface
  - egress-interface with IPv4 address
  - egress-interface with IPv6 address
  - egress-interface with MAC address
  - logical-tunnel
  - tunnel-encap
  - tunnel-decap
  - rib-name
- Chain: Provide a way to perform multiple operations on a packet by logically combining them.
- Load-balance: Designed for load-balance case where it normally will have multiple weighted nexthops.
- Protection: Designed for protection scenario where it normally will have primary and standby nexthop.
- Replicate: Designed for multiple destinations forwarding.

The structure tree of nexthop is shown in the following figures.

```
   +--rw nexthop
   |  +--rw nexthop-id?           uint32
   |  +--rw sharing-flag?         boolean
   |  +--rw (nexthop-type)?
   |     +--:(nexthop-base)
   |     |  ...(refer to Figure 5)
   |     +--:(nexthop-chain) {nexthop-chain}?
   |     |  +--rw nexthop-chain
   |     |     +--rw nexthop-list* [nexthop-member-id]
   |     |        +--rw nexthop-member-id uint32
   |     +--:(nexthop-replicates) {nexthop-replicates}?
   |     |  +--rw nexthop-replicates
   |     |     +--rw nexthop-list* [nexthop-member-id]
   |     |        +--rw nexthop-member-id uint32
   |     +--:(nexthop-protection) {nexthop-protection}?
   |     |  +--rw nexthop-protection
   |     |     +--rw nexthop-list* [nexthop-member-id]
   |     |        +--rw nexthop-member-id uint32
   |     |        +--rw nexthop-preference nexthop-preference-def
   |     +--:(nexthop-load-balance) {nexthop-load-balance}?
   |        +--rw nexthop-lb
   |           +--rw nexthop-list* [nexthop-member-id]
   |              +--rw nexthop-member-id uint32
   |              +--rw nexthop-lb-weight nexthop-lb-weight-def
```

Figure 4: Nexthop Structure

Figure 5 (as shown blow) is a sub-tree of nexthop, it’s under the nexthop base node and shows that structure of the "base" nexthop.

```
 +--:(nexthop-base)
 |  +--rw nexthop-base
 |     +--rw (nexthop-base-type)?
 |        +--:(special-nexthop)
 |        |  +--rw special? special-nexthop-def
 |        +--:(egress-interface-nexthop)
 |        |  +--rw outgoing-interface if:interface-ref
 |        +--:(ipv4-address-nexthop)
 |        |  +--rw ipv4-address inet:ipv4-address
 |        +--:(ipv6-address-nexthop)
 |        |  +--rw ipv6-address inet:ipv6-address
 |        +--:(egress-interface-ipv4-nexthop)
 |        |  +--rw egress-interface-ipv4-address
 |        |     +--rw outgoing-interface if:interface-ref
 |        |     +--rw ipv4-address       inet:ipv4-address
 |        +--:(egress-interface-ipv6-nexthop)
 |        |  +--rw egress-interface-ipv6-address
 |        |     +--rw outgoing-interface if:interface-ref
```

```
|        |     +--rw ipv6-address       inet:ipv6-address
|        +--:(egress-interface-mac-nexthop)
|        |  +--rw egress-interface-mac-address
|        |     +--rw outgoing-interface if:interface-ref
|        |     +--rw ieee-mac-address uint32
|        +--:(tunnel-encap-nexthop) {nexthop-tunnel}?
|        |  +--rw tunnel-encap
|        |     +--rw (tunnel-type)?
|        |        +--:(ipv4) {ipv4-tunnel}?
|        |        |  +--rw ipv4-header
|        |        |     +--rw src-ipv4-address inet:ipv4-address
|        |        |     +--rw dest-ipv4-address inet:ipv4-address
|        |        |     +--rw protocol          uint8
|        |        |     +--rw ttl?              uint8
|        |        |     +--rw dscp?             uint8
|        |        +--:(ipv6) {ipv6-tunnel}?
|        |        |  +--rw ipv6-header
|        |        |     +--rw src-ipv6-address inet:ipv6-address
|        |        |     +--rw dest-ipv6-address inet:ipv6-address
|        |        |     +--rw next-header       uint8
|        |        |     +--rw traffic-class? uint8
|        |        |     +--rw flow-label?       uint16
|        |        |     +--rw hop-limit?        uint8
|        |        +--:(mpls) {mpls-tunnel}?
|        |        |  +--rw mpls-header
|        |        |     +--rw label-operations* [label-oper-id]
|        |        |        +--rw label-oper-id uint32
|        |        |        +--rw (label-actions)?
|        |        |           +--:(label-push)
|        |        |           |  +--rw label-push
|        |        |           |     +--rw label        uint32
|        |        |           |     +--rw s-bit?       boolean
|        |        |           |     +--rw tc-value? uint8
|        |        |           |     +--rw ttl-value? uint8
|        |        |           +--:(label-swap)
|        |        |              +--rw label-swap
|        |        |                 +--rw in-label      uint32
|        |        |                 +--rw out-label     uint32
|        |        |                 +--rw ttl-action? ttl-action-def
|        |        +--:(gre) {gre-tunnel}?
|        |        |  +--rw gre-header
|        |        |     +--rw (dest-address-type)?
|        |        |     |  +--:(ipv4)
|        |        |     |  |  +--rw ipv4-dest inet:ipv4-address
|        |        |     |  +--:(ipv6)
|        |        |     |     +--rw ipv6-dest inet:ipv6-address
|        |        |     +--rw protocol-type uint16
|        |        |     +--rw key?          uint64
```

```
|        |        +--:(nvgre) {nvgre-tunnel}?
|        |        |  +--rw nvgre-header
|        |        |     +--rw (nvgre-type)?
|        |        |     |  +--:(ipv4)
|        |        |     |  |  +--rw src-ipv4-address inet:ipv4-address
|        |        |     |  |  +--rw dest-ipv4-address inet:ipv4-address
|        |        |     |  |  +--rw protocol          uint8
|        |        |     |  |  +--rw ttl?              uint8
|        |        |     |  |  +--rw dscp?             uint8
|        |        |     |  +--:(ipv6)
|        |        |     |     +--rw src-ipv6-address inet:ipv6-address
|        |        |     |     +--rw dest-ipv6-address inet:ipv6-address
|        |        |     |     +--rw next-header       uint8
|        |        |     |     +--rw traffic-class?    uint8
|        |        |     |     +--rw flow-label?       uint16
|        |        |     |     +--rw hop-limit?        uint8
|        |        |     +--rw virtual-subnet-id uint32
|        |        |     +--rw flow-id?          uint16
|        |        +--:(vxlan) {vxlan-tunnel}?
|        |           +--rw vxlan-header
|        |              +--rw (vxlan-type)?
|        |              |  +--:(ipv4)
|        |              |  |  +--rw src-ipv4-address inet:ipv4-address
|        |              |  |  +--rw dest-ipv4-address inet:ipv4-address
|        |              |  |  +--rw protocol             uint8
|        |              |  |  +--rw ttl?                 uint8
|        |              |  |  +--rw dscp?                uint8
|        |              |  +--:(ipv6)
|        |              |     +--rw src-ipv6-address inet:ipv6-address
|        |              |     +--rw dest-ipv6-address inet:ipv6-address
|        |              |     +--rw next-header          uint8
|        |              |     +--rw traffic-class?       uint8
|        |              |     +--rw flow-label?          uint16
|        |              |     +--rw hop-limit?           uint8
|        |              +--rw vxlan-identifier     uint32
|        +--:(tunnel-decap-nexthop) {nexthop-tunnel}?
|        |  +--rw tunnel-decap
|        |     +--rw (tunnel-type)?
|        |        +--:(ipv4) {ipv4-tunnel}?
|        |        |  +--rw ipv4-decap
|        |        |     +--rw ipv4-decap tunnel-decap-action-def
|        |        |     +--rw ttl-action?   ttl-action-def
|        |        +--:(ipv6) {ipv6-tunnel}?
|        |        |  +--rw ipv6-decap
|        |        |     +--rw ipv6-decap tunnel-decap-action-def
|        |        |     +--rw hop-limit-action? hop-limit-action-def
|        |        +--:(mpls) {mpls-tunnel}?
|        |           +--rw label-pop
```

```
|        |              +--rw label-pop     mpls-label-action-def
|        |              +--rw ttl-action?   ttl-action-def
|        +--:(logical-tunnel-nexthop) {nexthop-tunnel}?
|        |  +--rw logical-tunnel
|        |     +--rw tunnel-type tunnel-type-def
|        |     +--rw tunnel-name string
|        +--:(rib-name-nexthop)
|        |  +--rw rib-name?                        string
|        +--:(nexthop-identifier)
|           +--rw nexthop-ref                      nexthop-ref
```

Figure 5: Nexthop Base Structure

2.5. RPC Operations

This module defines the following RPC operations:

- rib-add: Add a RIB to a routing instance. A name of the rib, address family of the RIB and (optionally) whether the RPF check is enabled are passed as the input parameters. The output is the result of the add operation:
  - true - success;
  - false - failed; when failed, the i2rs agent may return the specific reason that causes the failure.
- rib-delete: Delete a RIB from a routing instance. When a RIB is deleted, all routes installed in the RIB will be deleted. A name of the RIB is passed as the input parameter. The output is the result of the delete operation:
  - true - success;
  - false - failed; when failed, the i2rs agent may return the specific reason that causes the failure.
- route-add: Add a route or a set of routes to a rib. A RIB name, the route prefix(es), route attributes, route vendor attributes, nexthop and whether return failure detail are passed as the input parameters. Before calling the route-add rpc, it is required to call the nh-add rpc to create and/or return the nexthop identifier. The output is a combination of the route operation states that include:
  - success-count: the number of routes that were successfully added;

    * failed-count: the number of the routes that failed to be added;

    * failure-detail: shows the specific routes that failed to be added.

- route-delete: Delete a route or a set of routes from a rib. A name of the rib, the route prefix(es) and whether to return failure detail are passed as the input parameters. The output is a combination of route operation states that include:

    * success-count: the number of routes that were successfully deleted;

    * failed-count: the number of the routes that failed to be deleted;

    * failure-detail: shows the specific routes that failed to be deleted.

- route-update: Update a route or a set of routes. A RIB name, the route prefix(es), or route attributes, or route vendor attributes, or nexthop are passed as the input parameters. The match conditions can be either route prefix(es), or route attributes, or route vendor attributes, or nexthop. The update actions include: update the nexthop, update the route attributes, update the route vendor attributes. The output is combination of the route operation states that include:

    * success-count: the number of routes that were successfully updated;

    * failed-count: the number of the routes that failed to be updated;

    * failure-detail: shows the specific routes that failed to be updated.

- nh-add: Add a nexthop to a rib. A name of the RIB and a nexthop are passed as the input parameters. The network node is required to allocate a nexthop identifier to the nexthop. The outputs include the result of the nexthop add operation.

    * true - success; when success, a nexthop identifier will be returned to the i2rs client.

    * false - failed; when failed, the i2rs agent may return the specific reason that causes the failure.

o nh-delete: Delete a nexthop from a rib. A name of a RIB and a nexthop or nexthop identifier are passed as the input parameters. The output is the result of the delete operation:

  * true - success;

  * false - failed; when failed, the i2rs agent may return the specific reason that causes the failure.

The structure tree of rpcs is shown in following figure.

```
rpcs:
   +---x rib-add
   |  +---w input
   |  |  +---w rib-name        string
   |  |  +---w address-family      rib-family-def
   |  |  +---w ip-rpf-check?   boolean
   |  +--ro output
   |     +--ro result uint32
   |     +--ro reason? string
   +---x rib-delete
   |  +---w input
   |  |  +---w rib-name string
   |  +--ro output
   |     +--ro result uint32
   |     +--ro reason? string
   +---x route-add
   |  +---w input
   |  |  +---w return-failure-detail?   boolean
   |  |  +---w rib-name                 string
   |  |  +---w routes
   |  |     +---w route-list* [route-index]
   |  |        ...
   |  +--ro output
   |     +--ro success-count     uint32
   |     +--ro failed-count      uint32
   |     +--ro failure-detail
   |        +--ro failed-routes* [route-index]
   |           +--ro route-index uint32
   |           +--ro error-code? uint32
   +---x route-delete
   |  +---w input
   |  |  +---w return-failure-detail?   boolean
   |  |  +---w rib-name                 string
   |  |  +---w routes
   |  |     +---w route-list* [route-index]
   |  |        ...
   |  +--ro output
```

```
|     +--ro success-count     uint32
|     +--ro failed-count      uint32
|     +--ro failure-detail
|        +--ro failed-routes* [route-index]
|           +--ro route-index uint32
|           +--ro error-code? uint32
+---x route-update
|  +---w input
|  |  +---w return-failure-detail?          boolean
|  |  +---w rib-name                        string
|  |  +---w (match-options)?
|  |     +--:(match-route-prefix)
|  |     |  ...
|  |     +--:(match-route-attributes)
|  |     |  ...
|  |     +--:(match-route-vendor-attributes) {...}?
|  |     |  ...
|  |     +--:(match-nexthop)
|  |        ...
|  +--ro output
|     +--ro success-count uint32
|     +--ro failed-count uint32
|     +--ro failure-detail
|        +--ro failed-routes* [route-index]
|           +--ro route-index uint32
|           +--ro error-code? uint32
+---x nh-add
|  +---w input
|  |  +---w rib-name              string
|  |  +---w nexthop-id?           uint32
|  |  +---w sharing-flag?         boolean
|  |  +---w (nexthop-type)?
|  |     ...
|  +--ro output
|     +--ro result        uint32
|     +--ro reason?       string
|     +--ro nexthop-id?   uint32
+---x nh-delete
   +---w input
   |  +---w rib-name              string
   |  +---w nexthop-id?           uint32
   |  +---w sharing-flag?         boolean
   |  +---w (nexthop-type)?
   |     ...
   +--ro output
      +--ro result uint32
      +--ro reason? string
```

Figure 6: RPCs Structure

2.6. Notifications

Asynchronous notifications are sent by the RIB manager of a network device to an external entity when some event triggers on the network device. An implementation of this RIB data model MUST support sending two kinds of asynchronous notifications.

1. Route change notification:

o Installed (Indicates whether the route got installed in the FIB) ;

o Active (Indicates whether a route has at least one fully resolved nexthop and is therefore eligible for installation in the FIB) ;

o Reason - E.g. Not authorized

2. Nexthop resolution status notification

Nexthops can be fully resolved or an unresolved.

A resolved nexthop has an adequate level of information to send the outgoing packet towards the destination by forwarding it on an interface to a directly connected neighbor.

An unresolved nexthop is something that requires the RIB manager to determine the final resolved nexthop. In one example, a nexthop could be an IP address. The RIB manager would resolve how to reach that IP address, e.g. by checking if that particular IP address is reachable by regular IP forwarding or by a MPLS tunnel or by both. If the RIB manager cannot resolve the nexthop, then the nexthop remains in an unresolved state and is NOT a suitable candidate for installation in the FIB.

An implementation of this RIB data model MUST support sending route-change notifications whenever a route transitions between the following states:

o from the active state to the inactive state

o from the inactive state to the active state

o from the installed state to the uninstalled state

o from the uninstalled state to the installed state

A single notification MAY be used when a route transitions from inactive/uninstalled to active/installed or in the other direction.

The structure tree of notifications is shown in the following figure.

```
notifications:
      +---n nexthop-resolution-status-change
      |  +--ro nexthop
      |  |  +--ro nexthop-id            uint32
      |  |  +--ro sharing-flag          boolean
      |  |  +--ro (nexthop-type)?
      |  |     +--:(nexthop-base)
      |  |     |  ...
      |  |     +--:(nexthop-chain) {nexthop-chain}?
      |  |     |  ...
      |  |     +--:(nexthop-replicates) {nexthop-replicates}?
      |  |     |  ...
      |  |     +--:(nexthop-protection) {nexthop-protection}?
      |  |     |  ...
      |  |     +--:(nexthop-load-balance) {nexthop-load-balance}?
      |  |        ...
      |  +--ro nexthop-state nexthop-state-def
      +---n route-change
         +--ro rib-name                string
         +--ro address-family              rib-family-def
         +--ro route-index             uint64
         +--ro match
         |  +--ro (route-type)?
         |     +--:(ipv4)
         |     |  ...
         |     +--:(ipv6)
         |     |  ...
         |     +--:(mpls-route)
         |     |  ...
         |     +--:(mac-route)
         |     |  ...
         |     +--:(interface-route)
         |        ...
         +--ro route-installed-state route-installed-state-def
         +--ro route-state             route-state-def
         +--ro route-change-reason     route-change-reason-def
```

Figure 7: Notifications Structure

3. YANG Modules

<CODE BEGINS> file "ietf-i2rs-rib@2016-07-04.yang"

```
module ietf-i2rs-rib {
  namespace "urn:ietf:params:xml:ns:yang:ietf-i2rs-rib";
  // replace with iana namespace when assigned
  prefix "iir";

  import ietf-inet-types {
    prefix inet;
    //rfc6991
  }

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF I2RS (Interface to Routing System) Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/i2rs/>
     WG List:  <mailto:i2rs@ietf.org>

     WG Chair: Susan Hares
               <mailto:shares@ndzh.com>

     WG Chair: Russ White
               <mailto:russ@riw.us>

     Editor:   Lixing Wang
               <mailto:wang_little_star@sina.com>

     Editor:   Hariharan Ananthakrishnan
               <mailto:hari@packetdesign.com>

     Editor:   Mach(Guoyi) Chen
               <mailto:mach.chen@huawei.com>

     Editor:   Amit Dass
               <mailto:amit.dass@ericsson.com>

     Editor:   Sriganesh Kini
               <mailto:sriganesh.kini@ericsson.com>
```

```
       Editor:   Nitin Bahadur
                 <mailto:nitin_bahadur@yahoo.com>";
    description
      "This module defines a YANG data model for
       Routing Information Base (RIB) that aligns
       with the I2RS RIB information model.";
    revision "2016-07-04" {
      description "initial revision";
      reference "draft-ietf-i2rs-data-model-06";
    }

    //Features
    feature nexthop-tunnel {
      description
        "This feature means that a node supports
         tunnel nexthop capability.";
    }

    feature nexthop-chain {
      description
        "This feature means that a node supports
         chain nexthop capability.";
    }

    feature nexthop-protection {
      description
        "This feature means that a node supports
         protection nexthop capability.";
    }

    feature nexthop-replicates {
      description
        "This feature means that a node supports
         replicates nexthop capability.";
    }

    feature nexthop-load-balance {
      description
        "This feature means that a node supports
         load balance nexthop capability.";
    }

    feature ipv4-tunnel {
      description
        "This feature means that a node supports
         IPv4 tunnel encapsulation capability.";
    }
```

```
feature ipv6-tunnel {
  description
    "This feature means that a node supports
     IPv6 tunnel encapsulation capability.";
}

feature mpls-tunnel {
  description
    "This feature means that a node supports
     MPLS tunnel encapsulation capability.";
}

feature vxlan-tunnel {
  description
    "This feature means that a node supports
     VxLAN tunnel encapsulation capability.";
}

feature gre-tunnel {
  description
    "This feature means that a node supports
     GRE tunnel encapsulation capability.";
}

feature nvgre-tunnel {
  description
    "This feature means that a node supports
     NvGRE tunnel encapsulation capability.";
}

feature route-vendor-attributes {
  description
    "This feature means that a node supports
     route vendor attributes.";
}

//Identities and Type Definitions
identity mpls-label-action {
  description
    "Base identity from which all MPLS label
     operations are derived.
     The MPLS label stack operations include:
     push - to add a new label to a label stack,
     pop - to pop the top label from a label stack,
     swap - to exchange the top label of a label
            stack with new label.";
}
```

```
identity label-push {
  base "mpls-label-action";
   description
    "MPLS label stack operation: push.";
}

identity label-pop {
  base "mpls-label-action";
  description
    "MPLS label stack operation: pop.";
}

identity label-swap {
  base "mpls-label-action";
  description
    "MPLS label stack operation: swap.";
}

typedef mpls-label-action-def {
  type identityref {
    base "mpls-label-action";
  }
  description
    "MPLS label action def.";
}

identity tunnel-decap-action {
  description
    "Base identity from which all tunnel decap
     actions are derived.
     Tunnel decap actions include:
     ipv4-decap - to decap an IPv4 tunnel,
     ipv6-decap - to decap an IPv6 tunnel.";
}

identity ipv4-decap {
  base "tunnel-decap-action";
  description
    "IPv4 tunnel decap.";
}

identity ipv6-decap {
  base "tunnel-decap-action";
  description
    "IPv4 tunnel decap.";
}

typedef tunnel-decap-action-def {
```

```
      type identityref {
        base "tunnel-decap-action";
      }
      description
        "Tunnel decap def.";
    }

    identity ttl-action {
      description
        "Base identity from which all TTL
         actions are derived.";
    }

    identity no-action {
      base "ttl-action";
      description
        "Do nothing regarding the TTL.";
    }

    identity copy-to-inner {
      base "ttl-action";
      description
        "Copy the TTL of the outer header
         to the inner header.";
    }

    identity decrease-and-copy-to-inner {
      base "ttl-action";
      description
        "Decrease TTL by one and copy the TTL
         to the inner header.";
    }

    identity decrease-and-copy-to-next {
      base "ttl-action";
      description
        "Decrease TTL by one and copy the TTL
         to the next header.For example: when
         MPLS label swapping, decrease the TTL
         of the inner label and copy it to the
         outer label.";
    }

    typedef ttl-action-def {
      type identityref {
        base "ttl-action";
      }
      description
```

```
        "TTL action def.";
    }

    identity hop-limit-action {
      description
        "Base identity from which all hop limit
         actions are derived.";
    }

    identity hop-limit-no-action {
      base "hop-limit-action";
      description
        "Do nothing regarding the hop limit.";
    }

    identity hop-limit-copy-to-inner {
      base "hop-limit-action";
      description
        "Copy the hop limit of the outer header
         to the inner header.";
    }

    typedef hop-limit-action-def {
      type identityref {
        base "hop-limit-action";
      }
      description
        "IPv6 hop limit action def.";
    }

    identity special-nexthop {
      description
        "Base identity from which all special
         nexthops are derived.";
    }

    identity discard {
      base "special-nexthop";
      description
        "This indicates that the network
         device should drop the packet and
         increment a drop counter.";
    }

    identity discard-with-error {
      base "special-nexthop";
      description
        "This indicates that the network
```

```
          device should drop the packet,
          increment a drop counter and send
          back an appropriate error message
          (like ICMP error).";
      }

      identity receive {
        base "special-nexthop";
        description
          "This indicates that the traffic is
           destined for the network device.  For
           example, protocol packets or OAM packets.
           All locally destined traffic SHOULD be
           throttled to avoid a denial of service
           attack on the router's control plane. An
           optional rate-limiter can be specified
           to indicate how to throttle traffic
           destined for the control plane.";
      }

      identity cos-value {
        base "special-nexthop";
        description
          "Cos-value special nexthop.";
      }

      typedef special-nexthop-def {
        type identityref {
          base "special-nexthop";
        }
        description
          "Special nexthop def.";
      }

      identity ip-route-match-type {
        description
          "Base identity from which all route
           match types are derived.
           Route match type could be:
           match source, or
           match destination, or
           match source and destination.";
      }

      identity match-ip-src {
        base "ip-route-match-type";
        description
          "Source route match type.";
```

```
  }
  identity match-ip-dest {
    base "ip-route-match-type";
    description
      "Destination route match type";
  }
  identity match-ip-src-dest {
    base "ip-route-match-type";
    description
      "Source and Destination route match type";
  }

  typedef ip-route-match-type-def {
    type identityref {
      base "ip-route-match-type";
    }
    description
      "IP route match type def.";
  }

  identity rib-family {
    description
      "Base identity from which all RIB
       address families are derived.";
  }

  identity ipv4-rib-family {
    base "rib-family";
    description
      "IPv4 RIB address family.";
  }

  identity ipv6-rib-family {
    base "rib-family";
    description
      "IPv6 RIB address family.";
  }

  identity mpls-rib-family {
    base "rib-family";
    description
      "MPLS RIB address family.";
  }

  identity ieee-mac-rib-family {
    base "rib-family";
    description
      "MAC RIB address family.";
```

```
     }

     typedef rib-family-def {
       type identityref {
         base "rib-family";
       }
       description
         "Rib address family def.";
     }

     identity route-type {
       description
         "Base identity from which all route types
          are derived.";
     }

     identity ipv4-route {
       base "route-type";
       description
         "IPv4 route type.";
     }

     identity ipv6-route {
       base "route-type";
       description
         "IPv6 route type.";
     }

     identity mpls-route {
       base "route-type";
       description
         "MPLS route type.";
     }

     identity ieee-mac {
       base "route-type";
       description
         "MAC route type.";
     }

     identity interface {
       base "route-type";
       description
         "Interface route type.";
     }

     typedef route-type-def {
       type identityref {
```

```
        base "route-type";
      }
      description
        "Route type def.";
    }

    identity tunnel-type {
      description
        "Base identity from which all tunnel
         types are derived.";
    }

    identity ipv4-tunnel {
      base "tunnel-type";
      description
        "IPv4 tunnel type";
    }

    identity ipv6-tunnel {
      base "tunnel-type";
      description
        "IPv6 Tunnel type";
    }

    identity mpls-tunnel {
      base "tunnel-type";
      description
        "MPLS tunnel type";
    }

    identity gre-tunnel {
      base "tunnel-type";
      description
        "GRE tunnel type";
    }

    identity vxlan-tunnel {
      base "tunnel-type";
      description
        "VxLAN tunnel type";
    }

    identity nvgre-tunnel {
      base "tunnel-type";
      description
        "NVGRE tunnel type";
    }
```

```
     typedef tunnel-type-def {
       type identityref {
         base "tunnel-type";
       }
       description
         "Tunnel type def.";
     }

     identity route-state {
       description
         "Base identity from which all route
          states are derived.";
     }

     identity active {
       base "route-state";
       description
         "Active state.";
     }

     identity inactive {
       base "route-state";
       description
         "Inactive state.";
     }

     typedef route-state-def {
       type identityref {
         base "route-state";
       }
       description
         "Route state def.";
     }

     identity nexthop-state {
       description
         "Base identity from which all nexthop
          states are derived.";
     }

     identity resolved {
       base "nexthop-state";
       description
         "Reolved nexthop state.";
     }

     identity unresolved {
       base "nexthop-state";
```

```
      description
        "Unresolved nexthop state.";
    }

    typedef nexthop-state-def {
      type identityref {
        base "nexthop-state";
      }
      description
        "Nexthop state def.";
    }

    identity route-installed-state {
      description
        "Base identity from which all route
         installed states are derived.";
    }

    identity uninstalled {
      base "route-installed-state";
      description
        "Uninstalled state.";
    }

    identity installed {
      base "route-installed-state";
      description
        "Installed state.";
    }

    typedef route-installed-state-def {
      type identityref {
        base "route-installed-state";
      }
      description
        "Route installed state def.";
    }

  //Route change reason identities

    identity route-change-reason {
      description
        "Base identity from which all route change
         reasons are derived.";
    }

    identity lower-route-preference {
      base "route-change-reason";
```

```
      description
        "This route was installed in the FIB because it had
         a lower route preference value (and thus was more
         preferred) than the route it replaced.";
    }

    identity higher-route-preference {
      base "route-change-reason";
      description
        "This route was uninstalled from the FIB because it had
         a higher route preference value (and thus was less
         preferred) than the route that replaced it.";
    }

    identity resolved-nexthop {
      base "route-change-reason";
      description
        "This route was made active because at least
         one of its nexthops was resolved.";
    }

    identity unresolved-nexthop {
      base "route-change-reason";
      description
        "This route was made inactive because all of
         its nexthops are unresolved.";
    }

    typedef route-change-reason-def {
      type identityref {
        base "route-change-reason";
      }
      description
        "Route change reason def.";
    }

    typedef nexthop-preference-def {
      type uint8 {
        range "1..99";
      }
      description
        "Nexthop-preference is used for protection schemes.
         It is an integer value between 1 and 99. Lower
         values are more preferred. To download N
         nexthops to the FIB, the N nexthops with the lowest
         value are selected. If there are more than N
         nexthops that have the same preference, an
         implementation of i2rs client should select N
```

```
          nexthops and download them, as for how to select
          the nexthops is left to the implementations.";
      }
      typedef nexthop-lb-weight-def {
        type uint8 {
          range "1..99";
        }
        description
          "Nexthop-lb-weight is used for load-balancing.
           Each list member MUST be assigned a weight
           between 1 and 99. The weight determines the
           proportion of traffic to be sent over a nexthop
           used for forwarding as a ratio of the weight of
           this nexthop divided by the weights of all the
           nexthops of this route that are used for forwarding.
           To perform equal load-balancing, one MAY specify
           a weight of 0 for all the member nexthops.  The
           value 0 is reserved for equal load-balancing
           and if applied, MUST be applied to all member nexthops.";
      }

      typedef nexthop-ref {
        type leafref {
          path  "/iir:routing-instance" +
                "/iir:rib-list" +
                "/iir:route-list" +
                "/iir:nexthop" +
                "/iir:nexthop-id";
        }
        description
          "A nexthop reference that provides
           an indirection reference to a nexthop.";
      }


      //Groupings
      grouping route-prefix {
        description
          "The common attributes used for all types of route prefix.";
        leaf route-index {
          type uint64 ;
          mandatory true;
          description
            "Route index.";
        }
        container match {
          description
            "The match condition specifies the
```

```
           kind of route (IPv4, MPLS, etc.)
           and the set of fields to match on.";
         choice route-type {
           description
            "Route types: IPv4, IPv6, MPLS, MAC etc.";
           case ipv4 {
             description
               "IPv4 route case.";
             container ipv4 {
               description
                 "IPv4 route match.";
               choice ip-route-match-type {
                 description
                   "IP route match type options:
                    match source, or
                    match destination, or
                    match source and destination.";
                 case dest-ipv4-address {
                   leaf dest-ipv4-prefix {
                     type inet:ipv4-prefix;
                     mandatory true;
                     description
                       "An IPv4 destination address as the match.";
                   }
                 }
                 case src-ipv4-address {
                   leaf src-ipv4-prefix {
                     type inet:ipv4-prefix;
                     mandatory true;
                     description
                       "An IPv4 source address as the match.";
                   }
                 }
                 case dest-src-ipv4-address {
                   container dest-src-ipv4-address {
                     description
                       "A combination of an IPv4 source and
                        an IPv4 destination address as the match.";
                     leaf dest-ipv4-prefix {
                       type inet:ipv4-prefix;
                       mandatory true;
                       description
                         "The IPv4 destination address of the match.";
                     }
                     leaf src-ipv4-prefix {
                       type inet:ipv4-prefix;
                       mandatory true;
                       description
```

```
                        "The IPv4 source address of the match";
                    }
                  }
                }
              }
            }
            case ipv6 {
              description
                "IPv6 route case.";
              container ipv6 {
                description
                  "IPv6 route match.";
                choice ip-route-match-type {
                  description
                    "IP route match type options:
                     match source, or
                     match destination, or
                     match source and destination.";
                  case dest-ipv6-address {
                    leaf dest-ipv6-prefix {
                      type inet:ipv6-prefix;
                      mandatory true;
                      description
                        "An IPv6 destination address as the match.";
                    }
                  }
                  case src-ipv6-address {
                    leaf src-ipv6-prefix {
                      type inet:ipv6-prefix;
                      mandatory true;
                      description
                        "An IPv6 source address as the match.";
                    }
                  }
                  case dest-src-ipv6-address {
                    container dest-src-ipv6-address {
                      description
                        "A combination of an IPv6 source and
                         an IPv6 destination address as the match.";
                      leaf dest-ipv6-prefix {
                        type inet:ipv6-prefix;
                        mandatory true;
                        description
                          "The IPv6 destination address of the match";
                      }
                      leaf src-ipv6-prefix {
                        type inet:ipv6-prefix;
```

```
                    mandatory true;
                    description
                      "The IPv6 source address of the match.";
                  }
                }
              }
            }
          }
          case mpls-route {
            description
              "MPLS route case.";
            leaf mpls-label {
              type uint32 ;
              mandatory true;
              description
                "The label used for matching.";
            }
          }
          case mac-route {
            description
              "MAC route case.";
            leaf mac-address {
              type uint32 ;
              mandatory true;
              description
                "The MAC address used for matching.";
            }
          }
          case interface-route {
            description
              "Interface route case.";
            leaf interface-identifier {
              type if:interface-ref;
              mandatory true;
              description
                "The interface used for matching.";
            }
          }
        }
      }
    }

    grouping route {
      description
        "The common attributes used for all types of routes.";
      uses route-prefix;
      container nexthop {
```

```
        description
          "The nexthop of the route.";
        uses nexthop;
      }
      //In the information model, it is called route-statistic
      container route-status {
        description
          "The status information of the route.";
        leaf route-state {
          type route-state-def;
          config false;
          description
            "Indicate a route's state: Active or Inactive.";
        }
        leaf route-installed-state {
          type route-installed-state-def;
          config false;
          description
            "Indicate that a route's installed states:
             Installed or uninstalled.";
        }
        leaf route-reason {
          type route-change-reason-def;
          config false;
          description
            "Indicate the reason that causes the route change.";
        }
      }
      container route-attributes {
        description
          "Route attributes.";
        uses route-attributes;
      }
      container route-vendor-attributes {
        description
          "Route vendor attributes.";
        uses route-vendor-attributes;
      }
    }

    grouping nexthop-list {
      description
        "A generic nexthop list.";
      list nexthop-list {
        key "nexthop-member-id";
        description
          "A list of nexthops.";
        leaf nexthop-member-id {
```

```
          type uint32;
          mandatory true;
          description
            "A nexthop identifier that points
             to a nexthop list member.
             A nexthop list member is a nexthop.";
        }
      }
    }

    grouping nexthop-list-p {
      description
        "A nexthop list with preference parameter.";
      list nexthop-list {
        key "nexthop-member-id";
        description
          "A list of nexthop.";
        leaf nexthop-member-id {
          type uint32;
          mandatory true;
          description
            "A nexthop identifier that points
             to a nexthop list member.
             A nexthop list member is a nexthop.";
        }
        leaf nexthop-preference {
          type nexthop-preference-def;
          mandatory true;
          description
            "Nexthop-preference is used for protection schemes.
             It is an integer value between 1 and 99.  Lower
             values are more preferred. To download a
             primary/standby/tertiary group to the FIB, the
             nexthops that are resolved and are most preferred
             are selected.";
        }
      }
    }

    grouping nexthop-list-w {
      description
        "A nexthop list with weight parameter.";
      list nexthop-list {
        key "nexthop-member-id";
        description
          "A list of nexthop.";
        leaf nexthop-member-id {
          type uint32;
```

```
          mandatory true;
          description
            "A nexthop identifier that points
             to a nexthop list member.
             A nexthop list member is a nexthop.";
        }
        leaf nexthop-lb-weight {
          type nexthop-lb-weight-def;
          mandatory true;
          description
            "The weight of a nexthop of
             the load balance nexthops.";
        }
      }
    }

    grouping nexthop {
      description
        "The nexthop structure.";
      leaf nexthop-id {
        type uint32;
        description
          "An identifier that refers to a nexthop.";
      }
      leaf sharing-flag {
        type boolean;
        description
          "To indicate whether a nexthop is sharable
           or non-sharable.
           true - sharable, means the nexthop can be shared
                  with other routes
           false - non-sharable, means the nexthop can not
                   be shared with other routes.";
      }
      choice nexthop-type {
        description
          "Nexthop type options.";
        case nexthop-base {
          container nexthop-base {
            description
              "The base nexthop.";
            uses nexthop-base;
          }
        }
        case nexthop-chain {
          if-feature nexthop-chain;
          container nexthop-chain {
            description
```

```
              "A chain nexthop.";
            uses nexthop-list;
          }
        }
        case nexthop-replicates {
          if-feature nexthop-replicates;
          container nexthop-replicates {
            description
              "A replicates nexthop.";
            uses nexthop-list;
          }
        }
        case nexthop-protection {
          if-feature nexthop-protection;
          container nexthop-protection {
            description
              "A protection nexthop.";
            uses nexthop-list-p;
          }
        }
        case nexthop-load-balance {
          if-feature nexthop-load-balance;
          container nexthop-lb {
            description
              "A load balance nexthop.";
            uses nexthop-list-w;
          }
        }
      }
    }

    grouping nexthop-base {
      description
        "The base nexthop.";
      choice nexthop-base-type {
        description
          "Nexthop base type options.";
        case special-nexthop {
          leaf special {
            type special-nexthop-def;
            description
              "A special nexthop.";
          }
        }
        case egress-interface-nexthop {
           leaf outgoing-interface {
             type if:interface-ref;
             mandatory true;
```

```
            description
              "The nexthop is an outgoing interface.";
           }
         }
         case ipv4-address-nexthop {
           leaf ipv4-address {
             type inet:ipv4-address;
             mandatory true;
             description
               "The nexthop is an IPv4 address.";
           }
         }
         case ipv6-address-nexthop {
           leaf ipv6-address {
             type inet:ipv6-address;
             mandatory true;
             description
               "The nexthop is an IPv6 address.";
           }
         }
         case egress-interface-ipv4-nexthop {
           container egress-interface-ipv4-address{
             leaf outgoing-interface {
               type if:interface-ref;
               mandatory true;
               description
                 "Name of the outgoing interface.";
             }
             leaf ipv4-address {
               type inet:ipv4-address;
               mandatory true;
               description
                 "The nexthop points to an interface with
                  an IPv4 address.";
             }
             description
               "The nexthop is an egress-interface and an IP
                address. This can be used in cases e.g. where
                the IP address is a link-local address.";
           }
         }
         case egress-interface-ipv6-nexthop {
           container egress-interface-ipv6-address {
             leaf outgoing-interface {
               type if:interface-ref;
               mandatory true;
               description
                 "Name of the outgoing interface.";
```

```
            }
            leaf ipv6-address {
              type inet:ipv6-address;
              mandatory true;
              description
                "The nexthop points to an interface with
                 an IPv6 address.";
            }
            description
              "The nexthop is an egress-interface and an IP
               address. This can be used in cases e.g. where
               the IP address is a link-local address.";
          }
        }
        case egress-interface-mac-nexthop {
          container egress-interface-mac-address {
            leaf outgoing-interface {
              type if:interface-ref;
              mandatory true;
              description
                "Name of the outgoing interface.";
            }
            leaf ieee-mac-address {
              type uint32;
              mandatory true;
              description
                "The nexthop points to an interface with
                 a specific mac-address.";
            }
            description
              "The egress interface must be an Ethernet
               interface. Address resolution is not required
               for this nexthop.";
          }
        }
        case tunnel-encap-nexthop {
          if-feature nexthop-tunnel;
          container tunnel-encap {
            uses tunnel-encap;
            description
              "This can be an encap representing an IP tunnel or
               MPLS tunnel or others as defined in info model.
               An optional egress interface can be chained to the
               tunnel encap to indicate which interface to send
               the packet out on.  The egress interface is useful
               when the network device contains Ethernet interfaces
               and one needs to perform address resolution for the
               IP packet.";
```

```
          }
        }
        case tunnel-decap-nexthop {
          if-feature nexthop-tunnel;
          container tunnel-decap {
            uses tunnel-decap;
            description
              "This is to specify decapsulating a tunnel header.";
          }
        }
        case logical-tunnel-nexthop {
          if-feature nexthop-tunnel;
          container logical-tunnel {
            uses logical-tunnel;
            description
              "This can be a MPLS LSP or a GRE tunnel (or others
               as defined in this document), that is represented
               by a unique identifier (e.g. name).";
          }
        }
        case rib-name-nexthop {
          leaf rib-name {
            type string;
            description
              "A nexthop pointing to a RIB indicates that the
               route lookup needs to continue in the specified
               rib. This is a way to perform chained lookups.";
          }
        }
        case nexthop-identifier {
          leaf nexthop-ref {
            type nexthop-ref;
            mandatory true;
            description
              "A nexthop reference that points to a nexthop.";
          }
        }
      }
    }

    grouping route-vendor-attributes {
      description
        "Route vendor attributes.";
    }

    grouping logical-tunnel {
      description
        "A logical tunnel that is identified
```

```
          by a type and a tunnel name.";
        leaf tunnel-type {
          type tunnel-type-def;
          mandatory true;
          description
            "A tunnel type.";
        }
        leaf tunnel-name {
          type string;
          mandatory true;
          description
            "A tunnel name that points to a logical tunnel.";
        }
      }

      grouping ipv4-header {
        description
          "The IPv4 header encapsulation information.";
        leaf src-ipv4-address {
          type inet:ipv4-address;
          mandatory true;
          description
            "The source IP address of the header.";
        }
        leaf dest-ipv4-address {
          type inet:ipv4-address;
          mandatory true;
          description
            "The destination IP address of the header.";
        }
        leaf protocol {
          type uint8;
          mandatory true;
          description
            "The protocol id of the header.";
        }
        leaf ttl {
          type uint8;
          description
            "The TTL of the header.";
        }
        leaf dscp {
          type uint8;
          description
            "The DSCP field of the header.";
        }
      }
```

```
    grouping ipv6-header {
      description
        "The IPv6 header encapsulation information.";
      leaf src-ipv6-address {
        type inet:ipv6-address;
        mandatory true;
        description
          "The source IP address of the header.";
      }
      leaf dest-ipv6-address {
        type inet:ipv6-address;
        mandatory true;
        description
          "The destination IP address of the header.";
      }
      leaf next-header {
        type uint8;
        mandatory true;
        description
          "The next header of the IPv6 header.";
      }
      leaf traffic-class {
        type uint8;
        description
          "The traffic class value of the header.";
      }
      leaf flow-label {
        type uint16;
        description
          "The flow label of the header.";
      }
      leaf hop-limit {
        type uint8;
        description
          "The hop limit the header.";
      }
    }

    grouping nvgre-header {
      description
        "The NvGRE header encapsulation information.";
      choice nvgre-type {
        description
          "NvGRE can use eigher IPv4
           or IPv6 header for encapsulation.";
        case ipv4 {
          uses ipv4-header;
        }
```

```
        case ipv6 {
          uses ipv6-header;
        }
      }
      leaf virtual-subnet-id {
        type uint32;
        mandatory true;
        description
          "The subnet identifier of the NvGRE header.";
      }
      leaf flow-id {
        type uint16;
        description
          "The flow identifier of the NvGRE header.";
      }
    }

    grouping vxlan-header {
      description
        "The VxLAN encapsulation header information.";
      choice vxlan-type {
        description
          "NvGRE can use either IPv4
           or IPv6 header for encapsulation.";
        case ipv4 {
          uses ipv4-header;
        }
        case ipv6 {
          uses ipv6-header;
        }
      }
      leaf vxlan-identifier {
        type uint32;
        mandatory true;
        description
          "The VxLAN identifier of the VxLAN header.";
      }
    }

    grouping gre-header {
      description
        "The GRE encapsulation header information.";
      choice dest-address-type {
        description
          "GRE options: IPv4 and IPv6";
        case ipv4 {
          leaf ipv4-dest {
            type inet:ipv4-address;
```

```
            mandatory true;
            description
              "The destination IP address of the GRE header.";
          }
        }
        case ipv6 {
          leaf ipv6-dest {
            type inet:ipv6-address;
            mandatory true;
            description
              "The destination IP address of the GRE header.";
          }
        }
      }
      leaf protocol-type {
        type uint16;
        mandatory true;
        description
          "The protocol type of the GRE header.";
      }
      leaf key {
        type uint64;
        description
          "The GRE key of the GRE header.";
      }
    }

    grouping mpls-header {
      description
        "The MPLS encapsulation header information.";
      list label-operations {
        key "label-oper-id";
        description
          "Label operations.";
        leaf label-oper-id {
          type uint32;
          description
            "An optional identifier that points
             to a label operation.";
        }
        choice label-actions {
          description
            "Label action options.";
          case label-push {
            container label-push {
              description
                "Label push operation.";
              leaf label {
```

```
                type uint32;
                mandatory true;
                description
                  "The label to be pushed.";
              }
              leaf s-bit {
                type boolean;
                description
                  "The s-bit of the label to be pushed. ";
              }
              leaf tc-value {
                type uint8;
                description
                  "The traffic class value of the label to be pushed.";
              }
              leaf ttl-value {
                type uint8;
                description
                  "The TTL value of the label to be pushed.";
              }
            }
          }
          case label-swap {
            container label-swap {
              description
                "Label swap operation.";
              leaf in-label {
                type uint32;
                mandatory true;
                description
                  "The label to be swapped.";
              }
              leaf out-label {
                type uint32;
                mandatory true;
                description
                  "The out MPLS label.";
              }
              leaf ttl-action {
                type ttl-action-def;
                description
                  "The label ttl actions:
                   - No-action, or
                   - Copy to inner label,or
                   - Decrease (the in label) by 1 and
                     copy to the out label.";
              }
            }
```

```
          }
        }
      }
    }

    grouping tunnel-encap{
      description
        "Tunnel encapsulation information.";
      choice tunnel-type {
        description
          "Tunnel options for next-hops.";
        case ipv4 {
          if-feature ipv4-tunnel;
          container ipv4-header {
            uses ipv4-header;
            description
              "IPv4 header.";
          }
        }
        case ipv6 {
          if-feature ipv6-tunnel;
          container ipv6-header {
            uses ipv6-header;
            description
              "IPv6 header.";
          }
        }
        case mpls {
          if-feature mpls-tunnel;
          container mpls-header {
            uses mpls-header;
            description
              "MPLS header.";
          }
        }
        case gre {
          if-feature gre-tunnel;
          container gre-header {
            uses gre-header;
            description
              "GRE header.";
          }
        }
        case nvgre {
          if-feature nvgre-tunnel;
          container nvgre-header {
            uses nvgre-header;
            description
```

```
              "NvGRE header.";
          }
        }
        case vxlan {
          if-feature vxlan-tunnel;
          container vxlan-header {
            uses vxlan-header;
            description
              "VxLAN header.";
          }
        }
      }
    }

    grouping tunnel-decap {
      description
        "Tunnel decapsulation information.";
      choice tunnel-type {
        description
          "Nexthop tunnel type options.";
        case ipv4 {
          if-feature ipv4-tunnel;
          container ipv4-decap {
            description
              "IPv4 decap.";
            leaf ipv4-decap  {
              type tunnel-decap-action-def;
              mandatory true;
              description
                "IPv4 decap operations.";
            }
            leaf ttl-action {
              type ttl-action-def;
              description
                "The ttl actions:
                 no-action or copy to inner header.";
            }
          }
        }
        case ipv6 {
          if-feature ipv6-tunnel;
          container ipv6-decap {
            description
              "IPv6 decap.";
            leaf ipv6-decap {
              type tunnel-decap-action-def;
              mandatory true;
              description
```

```
                "IPv6 decap operations.";
            }
            leaf hop-limit-action {
              type hop-limit-action-def;
              description
                "The hop limit actions:
                 no-action or copy to inner header.";
            }
          }
        }
        case mpls {
          if-feature mpls-tunnel;
          container label-pop {
            description
              "MPLS decap.";
            leaf label-pop {
              type mpls-label-action-def;
              mandatory true;
              description
                "Pop a label from the label stack.";
            }
            leaf ttl-action {
              type ttl-action-def;
              description
                "The label ttl action.";
            }
          }
        }
      }
    }

    grouping route-attributes {
      description
        "Route attributes.";
      leaf route-preference {
        type uint32;
        mandatory true;
        description
          "ROUTE_PREFERENCE: This is a numerical value that
           allows for comparing routes from different
           protocols.  Static configuration is also
           considered a protocol for the purpose of this
           field.  It is also known as administrative-distance.
           The lower the value, the higher the preference.";
      }
      leaf local-only {
        type boolean ;
        mandatory true;
```

```
        description
          "Indicate whether the attributes is local only.";
      }
      container address-family-route-attributes{
        description
          "Address family related route attributes.";
        choice route-type {
          description
            "Address family related route attributes.";
          case ip-route-attributes {
          }
          case mpls-route-attributes {
          }
          case ethernet-route-attributes {
          }
        }
      }
    }

    container routing-instance {
      description
        "A routing instance, in the context of
         the RIB information model, is a collection
         of RIBs, interfaces, and routing parameters";
      leaf name {
        type string;
        description
          "The name of the routing instance. This MUST
           be unique across all routing instances in
           a given network device.";
      }
      list interface-list {
        key "name";
        description
          "This represents the list of interfaces associated
           with this routing instance. The interface list helps
           constrain the boundaries of packet forwarding.
           Packets coming on these interfaces are directly
           associated with the given routing instance. The
           interface list contains a list of identifiers, with
           each identifier uniquely identifying an interface.";
        leaf name {
          type if:interface-ref;
          description
            "A reference to the name of a network layer interface.";
        }
      }
      leaf router-id {
```

```
        type yang:dotted-quad;
        description
          "Router ID - 32-bit number in the form of a dotted quad.";
      }
      leaf lookup-limit {
        type uint8;
        description
          "A limit on how many levels of a lookup can be performed.";
      }
      list rib-list {
        key "name";
        description
          "A list of RIBs that are associated with the routing
           instance.";
        leaf name {
          type string;
          mandatory true;
          description
            "A reference to the name of each rib.";
        }
        leaf address-family {
          type rib-family-def;
          mandatory true;
          description
            "The address family of a rib.";
        }
        leaf ip-rpf-check {
          type boolean;
          description
            "Each RIB can be optionally associated with a
             ENABLE_IP_RPF_CHECK attribute that enables Reverse
             path forwarding (RPF) checks on all IP routes in that
             RIB.  Reverse path forwarding (RPF) check is used to
             prevent spoofing and limit malicious traffic.";
        }
        list route-list {
          key "route-index";
          description
            "A list of routes of a rib.";
          uses route;
        }
        // This is a list that maintains the nexthops added to the rib.
        uses nexthop-list;
      }
    }

    //RPC Operations
    rpc rib-add {
```

```
      description
        "To add a RIB to a instance";
      input {
        leaf name {
          type string;
          mandatory true;
          description
            "A reference to the name of the RIB
             that is to be added.";
        }
        leaf address-family {
          type rib-family-def;
          mandatory true;
          description
            "The address family of the rib.";
        }
        leaf ip-rpf-check {
          type boolean;
          description
            "Each RIB can be optionally associated with a
             ENABLE_IP_RPF_CHECK attribute that enables Reverse
             path forwarding (RPF) checks on all IP routes in that
             RIB.  Reverse path forwarding (RPF) check is used to
             prevent spoofing and limit malicious traffic.";
        }
      }
      output {
        leaf result {
          type boolean;
          mandatory true;
          description
            "Return the result of the rib-add operation.
             true  - success;
             false - failed";
        }
        leaf reason {
          type string;
          description
            "The specific reason that causes the failure.";
        }
      }
    }

    rpc rib-delete {
      description
        "To delete a RIB from a routing instance.
         After deleting the rib, all routes installed
         in the RIB will be deleted as well.";
```

```
      input {
        leaf name {
          type string;
          mandatory true;
          description
            "A reference to the name of the RIB
             that is to be deleted.";
        }
      }
      output {
        leaf result {
          type boolean;
          mandatory true;
          description
            "Return the result of the rib-delete operation.
             true  - success;
             false - failed";
        }
        leaf reason {
          type string;
          description
            "The specific reason that causes failure.";
        }
      }
    }

    grouping route-operation-state {
      description
        "Route operation state.";
      leaf success-count {
        type uint32;
        mandatory true;
        description
          "The numbers of routes that are successfully
           added/deleted/updated.";
      }
      leaf failed-count {
        type uint32;
        mandatory true;
        description
          "The numbers of the routes that are failed
           to be added/deleted/updated.";
      }
      container failure-detail {
        description
          "The failure detail reflects the reason why a route
           operation fails. It is a array that includes the route
           index and error code of the failed route.";
```

```
        list failed-routes {
           key "route-index";
           description
            "The list of failed routes.";
           leaf route-index {
            type uint32;
            description
              "The route index of the failed route.";
          }
          leaf error-code {
            type uint32;
            description
              "The error code that reflects the failure reason.";
          }
        }
      }
    }

    rpc route-add {
      description
        "To add a route or a list of route to a rib";
      input {
        leaf return-failure-detail {
          type boolean;
          default false;
          description
            "Whether return the failure detail.
             true  - return the failure detail;
             false - do not return the failure detail;
             the default is false.";
        }
        leaf rib-name {
          type string;
          mandatory true;
          description
            "A reference to the name of a rib.";
        }
        container routes {
          description
            "The routes to be added to the rib.";
          list route-list {
            key "route-index";
            description
              "The list of routes to be added.";
            uses route-prefix;
            container route-attributes {
              uses route-attributes;
              description
```

```
                "The route attributes.";
            }
            container route-vendor-attributes {
              if-feature route-vendor-attributes;
              uses route-vendor-attributes;
              description
                "The route vendor attributes.";
            }
            container nexthop {
              uses nexthop;
              description
                "The nexthop of the added route.";
            }
          }
        }
      }
      output {
        uses route-operation-state;
      }
    }

    rpc route-delete {
      description
        "To delete a route or a list of route from a rib";
      input {
        leaf return-failure-detail {
          type boolean;
          default false;
          description
            "Whether return the failure detail.
             true  - return the failure detail;
             false - do not return the failure detail;
             the default is false.";
        }
        leaf rib-name {
          type string;
          mandatory true;
          description
            "A reference to the name of a rib.";
        }
        container routes {
          description
            "The routes to be added to the rib.";
          list route-list{
            key "route-index";
            description
              "The list of routes to be deleted.";
            uses route-prefix;
```

```
          }
        }
      }
      output {
        uses route-operation-state;
      }
    }

    grouping route-update-options {
      description
        "Update options:
         1. update the nexthop
         2. update the route attributes
         3. update the route-vendor-attributes.";
      choice update-options {
        description
          "Update options:
           1. update the nexthop
           2. update the route attributes
           3. update the route-vendor-attributes.";
        case update-nexthop {
          container updated-nexthop {
            uses nexthop;
            description
              "The nexthop used for updating.";
          }
        }
        case update-route-attributes {
          container updated-route-attr {
            uses route-attributes;
            description
              "The route attributes used for updating.";
          }
        }
        case update-route-vendor-attributes {
          container updated-route-vendor-attr {
            uses route-vendor-attributes;
            description
              "The vendor route attributes used for updating.";
          }
        }
      }
    }

    rpc route-update {
      description
        "To update a route or a list of route of a rib.
         The inputs:
```

```
            1. The match conditions, could be:
              a. route prefix, or
              b. route attributes, or
              c. nexthop;
            2. The update parameters to be used:
              a. new nexthop;
              b. new route attributes;nexthop
           Actions:
            1. update the nexthop
            2. update the route attributes
           The outputs:
            success-count - the number of routes updated;
            failed-count - the number of routes fail to update
            failure-detail - the detail failure info.
           ";
        input {
          leaf return-failure-detail {
            type boolean;
            default false;
            description
              "Whether return the failure detail.
               true  - return the failure detail;
               false - do not return the failure detail;
               the default is false.";
          }
          leaf rib-name {
            type string;
            mandatory true;
            description
              "A reference to the name of a rib.";
          }
          choice match-options {
            description
              "Match options.";
            case match-route-prefix {
              description
                "Update the routes that match route
                 prefix(es) condition.";
              container input-routes {
                description
                  "The matched routes to be updated.";
                list route-list {
                  key "route-index";
                  description
                    "The list of routes to be updated.";
                  uses route-prefix;
                  uses route-update-options;
                }
```

```
            }
          }
          case match-route-attributes {
            description
              "Update the routes that match the
               route attributes condition.";
            container input-route-attributes {
              description
                "The route attributes are used for matching.";
              uses route-attributes;
            }
            container update-parametors {
              description
                "Update options:
                 1. update the nexthop
                 2. update the route attributes
                 3. update the route-vendor-attributes.";
              uses route-update-options;
            }
          }
          case match-route-vendor-attributes {
            if-feature route-vendor-attributes;
            description
              "Update the routes that match the
               vendor attributes condition";
            container input-route-vendor-attributes {
              description
                "The vendor route attributes are used for matching.";
              uses route-vendor-attributes;
            }
            container update-parameters-vendor {
              description
                "Update options:
                 1. update the nexthop
                 2. update the route attributes
                 3. update the route-vendor-attributes.";
              uses route-update-options;
            }
          }
          case match-nexthop {
            description
              "Update the routes that match the nexthop.";
            container input-nexthop {
              description
                "The nexthop used for matching.";
              uses nexthop;
            }
            container update-parameters-nexthop {
```

```
              description
                "Update options:
                 1. update the nexthop
                 2. update the route attributes
                 3. update the route-vendor-attributes.";
              uses route-update-options;
            }
          }
        }
      }
      output {
        uses route-operation-state;
      }
    }

    rpc nh-add {
      description
        "To add a nexthop to a rib.
         Inputs parameters:
           1. RIB name
           2. nexthop;
         Actions:
           Add the nexthop to the RIB
         Outputs:
           1.Operation result:
             true  - success
             false - failed;
           2. nexthop identifier.";
      input {
        leaf rib-name {
          type string;
          mandatory true;
          description
            "A reference to the name of a rib.";
        }
        uses nexthop;
      }
      output {
        leaf result {
          type boolean;
          mandatory true;
          description
            "Return the result of the rib-add operation.
             true  - success;
             false - failed;";
        }
        leaf reason {
          type string;
```

```
          description
            "The specific reason that causes the failure.";
        }
        leaf nexthop-id {
          type uint32;
          description
            "A nexthop identifier that is allocated to the nexthop.";
        }
      }
    }

    rpc nh-delete {
      description
        "To delete a nexthop from a rib";
      input {
        leaf rib-name {
          type string;
          mandatory true;
          description
            "A reference to the name of a rib.";
        }
        uses nexthop;
      }
      output {
        leaf result {
          type boolean;
          mandatory true;
          description
            "Return the result of the rib-add operation.
             true  - success;
             false - failed.";
        }
        leaf reason {
          type string;
          description
            "The specific reason that causes the failure.";
        }
      }
    }

    /*Notifications*/
    notification nexthop-resolution-status-change {
      description
        "Nexthop resolution status (resolved/unresolved)
         notification.";
      container nexthop{
        description
          "The nexthop.";
```

```
        uses nexthop;
      }
      leaf nexthop-state {
        type nexthop-state-def;
        mandatory true;
        description
          "Nexthop resolution status (resolved/unresolved)
           notification.";
      }
    }

    notification route-change {
      description
        "Route change notification.";
      leaf rib-name {
        type string;
        mandatory true;
        description
          "A reference to the name of a rib.";
      }
      leaf address-family {
        type rib-family-def;
        mandatory true;
        description
          "The address family of a rib.";
      }
      uses route-prefix;
      leaf route-installed-state {
        type route-installed-state-def;
        mandatory true;
        description
          "Indicates whether the route got installed in the FIB.";
      }
      leaf route-state {
        type route-state-def;
        mandatory true;
        description
          "Indicates whether a route is active or inactive.";
      }
      list route-change-reasons {
        key "route-change-reason";
        description
          "The reasons that cause the route change. A route
           change that may result from several reasons. For
           example, a nexthop becoming resolved will make a
           route A active which is of better preference than
           a currently active route B, which results in the
           route A being installed";
```

```
        leaf route-change-reason {
          type route-change-reason-def;
          mandatory true;
          description
            "The reason that causes the route change.";
        }
      }
    }
  }
```

<CODE ENDS>

4. IANA Considerations

This document requests to register a URI in the "ns" registry with the "IETF XML registry" [RFC3688]:

```
---------------------------------------------------------------------
URI: urn:ietf:params:xml:ns:yang:ietf-i2rs-rib
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.
---------------------------------------------------------------------
```

This document requests to register a YANG module in the "YANG Module Names registry" [RFC6020]:

```
---------------------------------------------------------------------
name:         ietf-i2rs-rib
namespace:    urn:ietf:params:xml:ns:yang:ietf-i2rs-rib
prefix:       iir
reference:    RFC XXXX
---------------------------------------------------------------------
```

5. Security Considerations

I2RS protocol provides read and write access to the information and state (e.g., RIB) within the routing process that exists inside the routing elements. These information and state are normally considered sensitive or vulnerable. Improper write operations to these information and state can have negative effects on the network.

The I2RS protocol will provide security mechanisms as required in [I-D.ietf-i2rs-security-environment-reqs] and [I-D.ietf-i2rs-protocol-security-requirements].

The YANG data model defined in this document itself will not introduce extra security issues.

6. Contributors

The following individuals also contribute to this document.

- Zekun He, Tencent Holdings Ltd
- Sujian Lu, Tencent Holdings Ltd
- Jeffery Zhang, Juniper Networks

7. Acknowledgements

The authors would like to thank Chris Bowers and John Scudder for his review, suggestion and comments to this document.

8. References

8.1. Normative References

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

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

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

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

8.2. Informative References

[I-D.ietf-i2rs-architecture] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-15 (work in progress), April 2016.

[I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-06 (work in progress), May 2016.

[I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-08 (work in progress), October 2015.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-01 (work in progress), April 2016.

[I-D.ietf-i2rs-usecase-reqs-summary]
Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", draft-ietf-i2rs-usecase-reqs-summary-02 (work in progress), March 2016.

Authors' Addresses

Lixing Wang
Individual

Email: wang_little_star@sina.com

Hariharan Ananthakrishnan
Packet Design

Email: hari@packetdesign.com

Mach(Guoyi) Chen
Huawei

Email: mach.chen@huawei.com

Amit Dass
Ericsson
Torshamnsgatan 48.
Stockholm 16480
Sweden

Email: amit.dass@ericsson.com

Sriganesh Kini
Ericsson

Email: sriganesh.kini@ericsson.com

Nitin Bahadur
Bracket Computing

Email: nitin_bahadur@yahoo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 8, 2017

N. Bahadur, Ed.
Bracket Computing
S. Kini, Ed.

J. Medved
Cisco

July 7, 2016

Routing Information Base Info Model
draft-ietf-i2rs-rib-info-model-09

Abstract

Routing and routing functions in enterprise and carrier networks are typically performed by network devices (routers and switches) using a routing information base (RIB). Protocols and configuration push data into the RIB and the RIB manager installs state into the hardware; for packet forwarding. This draft specifies a information model for the RIB to enable defining a standardized data model. Such a data model can be used to define an interface to the RIB from an entity that may even be external to the network device. This interface can be used to support new use-cases being defined by the IETF I2RS WG.

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 January 8, 2017.

Table of Contents

1. Introduction

Routing and routing functions in enterprise and carrier networks are traditionally performed in network devices. Traditionally routers run routing protocols and the routing protocols (along with static config) populate the Routing information base (RIB) of the router. The RIB is managed by the RIB manager and the RIB manager provides a north-bound interface to its clients i.e. the routing protocols to insert routes into the RIB. The RIB manager consults the RIB and decides how to program the forwarding information base (FIB) of the hardware by interfacing with the FIB manager. The relationship between these entities is shown in Figure 1.

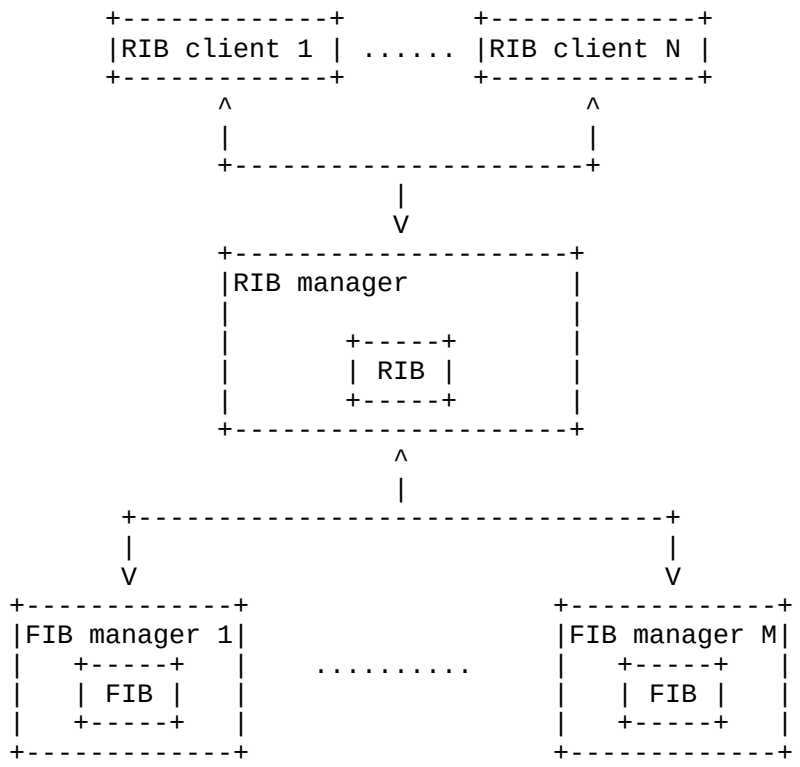


Figure 1: RIB manager, RIB clients and FIB managers

Routing protocols are inherently distributed in nature and each router makes an independent decision based on the routing data received from its peers. With the advent of newer deployment paradigms and the need for specialized applications, there is an emerging need to guide the router’s routing function [RFC7920]. Traditional network-device protocol-based RIB population suffices for

most use cases where distributed network control is used. However there are use cases which the network operators currently address by configuring static routes, policies and RIB import/export rules on the routers. There is also a growing list of use cases [I-D.white-i2rs-use-case], [I-D.hares-i2rs-use-case-vn-vc] in which a network operator might want to program the RIB based on data unrelated to just routing (within that network's domain). Programming the RIB could be based on other information such as routing data in the adjacent domain or the load on storage and compute in the given domain. Or it could simply be a programmatic way of creating on-demand dynamic overlays (e.g. GRE tunnels) between compute hosts (without requiring the hosts to run traditional routing protocols). If there was a standardized publicly documented programmatic interface to a RIB, it would enable further networking applications that address a variety of use-cases [RFC7920].

A programmatic interface to the RIB involves 2 types of operations - reading from the RIB and writing (adding/modifying/deleting) to the RIB. [I-D.white-i2rs-use-case] lists various use-cases which require read and/or write manipulation of the RIB.

In order to understand what is in a router's RIB, methods like per-protocol SNMP MIBs and show output screen scraping are used. These methods are not scalable, since they are client pull mechanisms and not proactive push (from the router) mechanisms. Screen scraping is error prone (since the output format can change) and is vendor dependent. Building a RIB from per-protocol MIBs is error prone since the MIB data represent protocol data and not the exact information that went into the RIB. Thus, just getting read-only RIB information from a router is a hard task.

Adding content to the RIB from an external entity can be done today using static configuration mechanisms provided by router vendors. However the mix of what can be modified in the RIB varies from vendor to vendor and the method of configuring it is also vendor dependent. This makes it hard for an external entity to program a multi-vendor network in a consistent and vendor-independent way.

The purpose of this draft is to specify an information model for the RIB. Using the information model, one can build a detailed data model for the RIB. That data model could then be used by an external entity to program a network device.

The rest of this document is organized as follows. Section 2 goes into the details of what constitutes and can be programmed in a RIB. Guidelines for reading and writing the RIB are provided in Section 3 and Section 4 respectively. Section 5 provides a high-level view of the events and notifications going from a network device to an

external entity, to update the external entity on asynchronous events. The RIB grammar is specified in Section 6. Examples of using the RIB grammar are shown in Section 7. Section 8 covers considerations for performing RIB operations at scale.

1.1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. RIB data

This section describes the details of a RIB. It makes forward references to objects in the RIB grammar (Section 6). A high-level description of the RIB contents is as shown below.

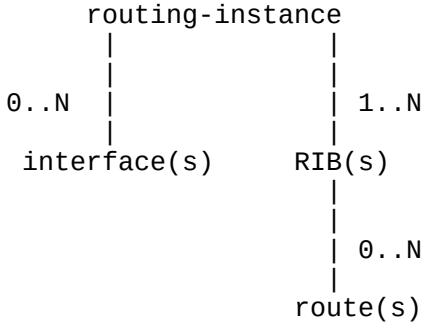


Figure 2: RIB model

2.1. RIB definition

A RIB is an entity that contains routes. A RIB is identified by its name and a RIB is contained within a routing instance (Section 2.2). The name MUST be unique within a routing instance. All routes in a given RIB MUST be of the same rib family (e.g. IPv4). Each RIB MUST belong to a routing instance.

A routing instance can have multiple RIBs. A routing instance can even have two or more RIBs of the same rib family (e.g. IPv6). A typical case where this can be used is for multi-topology routing ([RFC4915], [RFC5120]).

Each RIB can be optionally associated with a ENABLE_IP_RPF_CHECK attribute that enables Reverse path forwarding (RPF) checks on all IP routes in that RIB. Reverse path forwarding (RPF) check is used to

prevent spoofing and limit malicious traffic. For IP packets, the IP source address is looked up and the rpf interface(s) associated with the route for that IP source address is found. If the incoming IP packet's interface matches one of the rpf interface(s), then the IP packet is forwarded based on its IP destination address; otherwise, the IP packet is discarded.

2.2. Routing instance

A routing instance, in the context of the RIB information model, is a collection of RIBs, interfaces, and routing parameters. A routing instance creates a logical slice of the router. It allows different logical slices; across a set of routers; to communicate with each other. Layer 3 Virtual Private Networks (VPN), Layer 2 VPNs (L2VPN) and Virtual Private Lan Service (VPLS) can be modeled as routing instances. Note that modeling a Layer 2 VPN using a routing instance only models the Layer-3 (RIB) aspect and does not model any layer-2 information (like ARP) that might be associated with the L2VPN.

The set of interfaces indicates which interfaces are associated with this routing instance. The RIBs specify how incoming traffic is to be forwarded. And the routing parameters control the information in the RIBs. The intersection set of interfaces of 2 routing instances MUST be the null set. In other words, an interface MUST NOT be present in 2 routing instances. Thus a routing instance describes the routing information and parameters across a set of interfaces.

A routing instance MUST contain the following mandatory fields.

- INSTANCE_NAME: A routing instance is identified by its name, INSTANCE_NAME. This MUST be unique across all routing instances in a given network device.
- rib-list: This is the list of RIBs associated with this routing instance. Each routing instance can have multiple RIBs to represent routes of different types. For example, one would put IPv4 routes in one RIB and MPLS routes in another RIB.

A routing instance MAY contain the following optional fields.

- interface-list: This represents the list of interfaces associated with this routing instance. The interface list helps constrain the boundaries of packet forwarding. Packets coming on these interfaces are directly associated with the given routing instance. The interface list contains a list of identifiers, with each identifier uniquely identifying an interface.
- ROUTER_ID: The router-id field identifies the network device in control plane interactions with other network devices. This field is to be used if one wants to virtualize a physical router into multiple virtual routers. Each virtual router MUST have a unique router-id. ROUTER_ID MUST be unique across all network devices in

a given domain.
A routing instance may be created purely for the purposes of packet processing and may not have any interfaces associated with it. For example, an incoming packet in routing instance A might have a nexthop of routing instance B and after packet processing in B, the nexthop might be routing instance C. Thus, routing instance B is not associated with any interface. And given that this routing instance does not do any control plane interaction with other network devices, a ROUTER_ID is also not needed.

2.3. Route

A route is essentially a match condition and an action following the match. The match condition specifies the kind of route (IPv4, MPLS, etc.) and the set of fields to match on. Figure 3 represents the overall contents of a route.

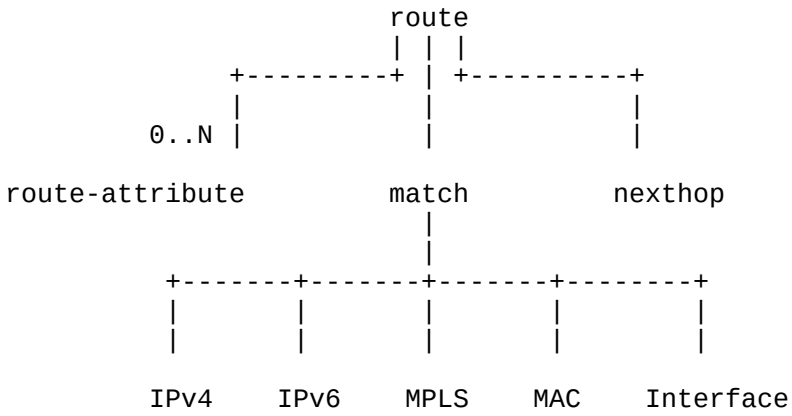


Figure 3: Route model

This document specifies the following match types:
- IPv4: Match on destination IP address in the IPv4 header
- IPv6: Match on destination IP address in the IPv6 header
- MPLS: Match on a MPLS label at the top of the MPLS label stack
- MAC: Match on MAC destination addresses in the ethernet header
- Interface: Match on incoming interface of the packet
- IP multicast: Match on (S, G) or (*, G), where S and G are IP addresses

Each route MUST have associated with it the following mandatory route

attributes.

- ROUTE_PREFERENCE: This is a numerical value that allows for comparing routes from different protocols. Static configuration is also considered a protocol for the purpose of this field. It is also known as administrative-distance. The lower the value, the higher the preference. For example there can be an OSPF route for 192.0.2.1/32 (or IPv6 2001:DB8::1/32) with a preference of 5. If a controller programs a route for 192.0.2.1/32 (or IPv6 2001: DB8::1/32) with a preference of 2, then the controller's route will be preferred by the RIB manager. Preference should be used to dictate behavior. For more examples of preference, see Section 7.1.

Each route can have associated with it one or more optional route attributes.

- route-vendor-attributes: Vendors can specify vendor-specific attributes using this. The details of this attribute is outside the scope of this document.

2.4. Nexthop

A nexthop represents an object resulting from a route lookup. For example, if a route lookup results in sending the packet out a given interface, then the nexthop represents that interface.

Nexthops can be fully resolved nexthops or unresolved nexthop. A resolved nexthop has adequate information to send the outgoing packet to the destination by forwarding it on an interface to a directly connected neighbor. For example, a nexthop to a point-to-point interface or a nexthop to an IP address on an Ethernet interface has the nexthop resolved. An unresolved nexthop is something that requires the RIB manager to determine the final resolved nexthop. For example, a nexthop could be an IP address. The RIB manager would resolve how to reach that IP address, e.g. is the IP address reachable by regular IP forwarding or by a MPLS tunnel or by both. If the RIB manager cannot resolve the nexthop, then the nexthop remains in an unresolved state and is NOT a candidate for installation in the FIB. Future RIB events can cause an unresolved nexthop to get resolved (like that IP address being advertised by an IGP neighbor). Conversely resolved nexthops can also become unresolved (e.g. in case of a tunnel going down) and hence would no longer be candidates to be installed in the FIB.

When at least one of a route's nexthops is resolved, then the route can be used to forward packets. Such a route is considered eligible to be installed in the FIB and is henceforth referred to as a FIB-eligible route. Conversely, when all the nexthops of a route are unresolved that route can no longer be used to forward packets. Such

a route is considered ineligible to be installed in the FIB and is henceforth referred to as a FIB-ineligible route. The RIB information model allows an external entity to program routes whose nexthops may be unresolved initially. Whenever an unresolved nexthop gets resolved, the RIB manager will send a notification of the same (see Section 5).

The overall structure and usage of a nexthop is as shown in the figure below.

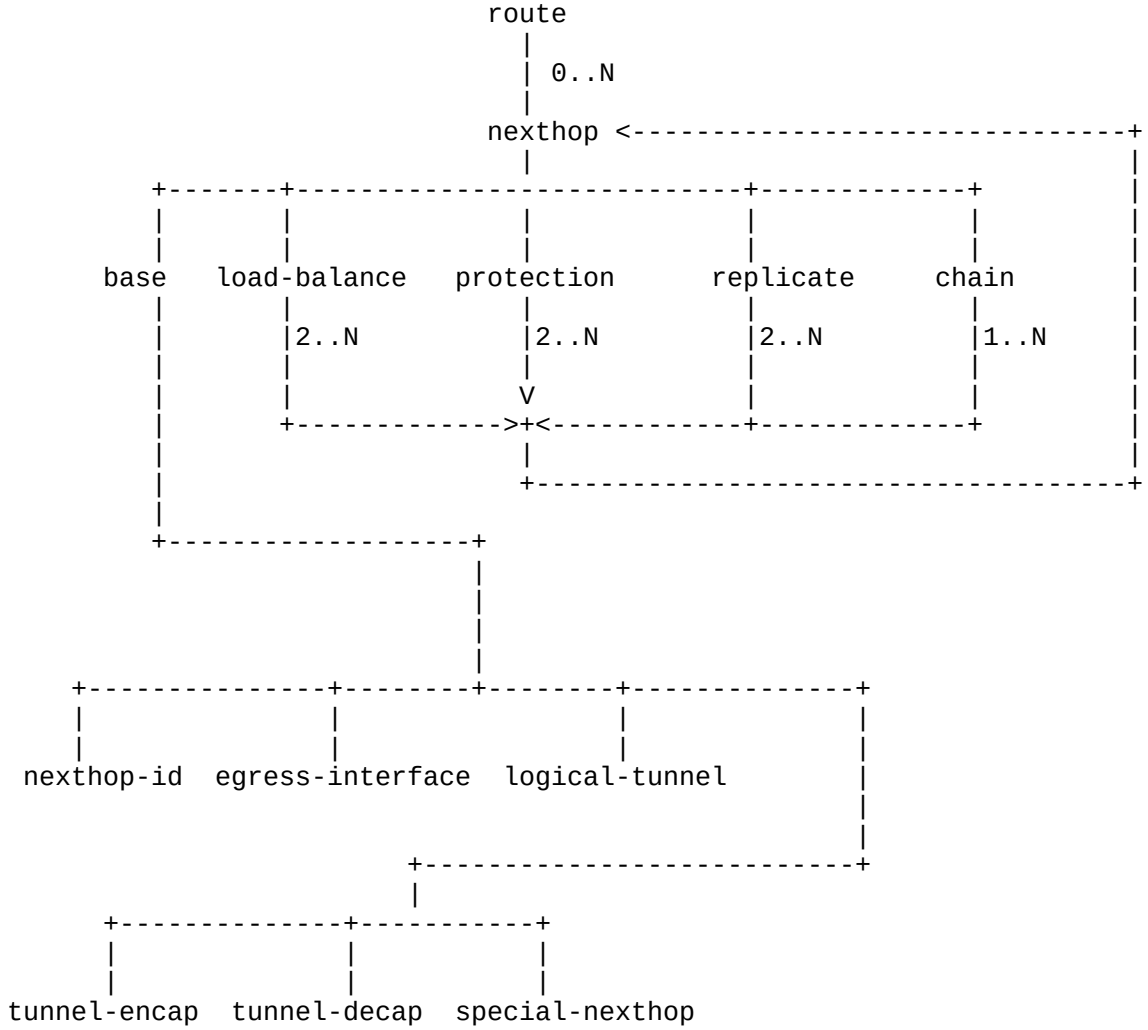


Figure 4: Nexthop model

Nexthops can be identified by an identifier to create a level of indirection. The identifier is set by the RIB manager and returned to the external entity on request. The RIB data-model SHOULD support a way to optionally receive a nexthop identifier for a given nexthop. For example, one can create a nexthop that points to a BGP peer. The returned nexthop identifier can then be used for programming routes to point to the same nexthop. Given that the RIB manager has created an indirection for that BGP peer using the nexthop identifier, if the transport path to the BGP peer changes, that change in path will be seamless to the external entity and all routes that point to that BGP peer will automatically start going over the new transport path. Nexthop indirection using identifiers could be applied to not just unicast nexthops, but even to nexthops that contain chains and nested nexthops (Section 2.4.1).

2.4.1. Nexthop types

This document specifies a very generic, extensible and recursive grammar for nexthops. Nexthops can be

- Interface nexthops - pointing to an interface
- Tunnel nexthops - pointing to a tunnel
- Replication lists - list of nexthops to which to replicate a packet
- Weighted lists - for load-balancing
- Preference lists - for protection using primary and backup
- Nexthop chains - for chaining multiple operations or attaching multiple headers
- Lists of lists - recursive application of the above
- Indirect nexthops - pointing to a nexthop identifier
- Special nexthops - for performing specific well-defined functions (e.g. drop)

It is expected that all network devices will have a limit on how many levels of lookup can be performed and not all hardware will be able to support all kinds of nexthops. RIB capability negotiation becomes very important for this reason and a RIB data-model MUST specify a way for an external entity to learn about the network device's capabilities. Examples of when and how to use various kinds of nexthops are shown in Section 7.2.

Tunnel nexthops allow an external entity to program static tunnel headers. There can be cases where the remote tunnel end-point does not support dynamic signaling (e.g. no LDP support on a host) and in those cases the external entity might want to program the tunnel header on both ends of the tunnel. The tunnel nexthop is kept generic with specifications provided for some commonly used tunnels. It is expected that the data-model will model these tunnel types with complete accuracy.

Nexthop chains Section 7.2.5, is a way to perform multiple operations on a packet by logically combining them. For example, one can chain together "decapsulate MPLS header" and "send it out a specific EGRESS_INTERFACE". Chains can be used to specify multiple headers over a packet, before a packet is forwarded. One simple example is that of MPLS over GRE, wherein the packet has an inner MPLS header followed by a GRE header followed by an IP header. The outermost IP header is decided by the network device whereas the MPLS header and GRE header are specified by the controller. Not every network device will be able to support all kinds of nexthop chains and an arbitrary number of header chained together. The RIB data-model SHOULD provide a way to expose nexthop chaining capability supported by a given network device.

2.4.2. Nexthop list attributes

For nexthops that are of the form of a list(s), attributes can be associated with each member of the list to indicate the role of an individual member of the list. Two attributes are specified:

- NEXTHOP_PREFERENCE: This is used for protection schemes. It is an integer value between 1 and 99. A lower value indicates higher preference. To download a primary/standby pair to the FIB, the nexthops that are resolved and have two highest preferences are selected. Each <NEXTHOP_PREFERENCE> should have a unique value within a <nexthop-protection>
  *
  (Section 6).
- NEXTHOP_LB_WEIGHT: This is used for load-balancing. Each list member MUST be assigned a weight between 1 and 99. The weight determines the proportion of traffic to be sent over a nexthop used for forwarding as a ratio of the weight of this nexthop divided by the weights of all the nexthops of this route that are used for forwarding. To perform equal load-balancing, one MAY specify a weight of "0" for all the member nexthops. The value "0" is reserved for equal load-balancing and if applied, MUST be applied to all member nexthops.

2.4.3. Nexthop content

At the lowest level, a nexthop can be one of:

- identifier: This is an identifier returned by the network device representing a nexthop. This can be used as a way of re-using a nexthop when programming complex nexthops.
- EGRESS_INTERFACE: This represents a physical, logical or virtual interface on the network device. Address resolution must not be required on this interface. This interface may belong to any routing instance.

- IP address: A route lookup on this IP address is done to determine the egress interface. Address resolution may be required depending on the interface.
  - An optional RIB name can also be specified to indicate the RIB in which the IP address is to be looked up. One can use the RIB name field to direct the packet from one domain into another domain. By default the RIB will be the same as the one that route belongs to.
- EGRESS_INTERFACE and IP address: This can be used in cases e.g. where the IP address is a link-local address.
- EGRESS_INTERFACE and MAC address: The egress interface must be an ethernet interface. Address resolution is not required for this nexthop.
- tunnel encap: This can be an encap representing an IP tunnel or MPLS tunnel or others as defined in this document. An optional egress interface can be chained to the tunnel encap to indicate which interface to send the packet out on. The egress interface is useful when the network device contains Ethernet interfaces and one needs to perform address resolution for the IP packet.
- tunnel decap: This is to specify decapsulating a tunnel header. After decap, further lookup on the packet can be done via chaining it with another nexthop. The packet can also be sent out via a EGRESS_INTERFACE directly.
- logical tunnel: This can be a MPLS LSP or a GRE tunnel (or others as defined in this document), that is represented by a unique identifier (E.g. name).
- RIB_NAME: A nexthop pointing to a RIB indicates that the route lookup needs to continue in the specified RIB. This is a way to perform chained lookups.

2.4.4. Special nexthops

This document specifies certain special nexthops. The purpose of each of them is explained below:

- DISCARD: This indicates that the network device should drop the packet and increment a drop counter.
- DISCARD_WITH_ERROR: This indicates that the network device should drop the packet, increment a drop counter and send back an appropriate error message (like ICMP error).
- RECEIVE: This indicates that that the traffic is destined for the network device. For example, protocol packets or OAM packets. All locally destined traffic SHOULD be throttled to avoid a denial of service attack on the router's control plane. An optional rate-limiter can be specified to indicate how to throttle traffic destined for the control plane. The description of the rate-limiter is outside the scope of this document.

3. Reading from the RIB

A RIB data-model MUST allow an external entity to read entries, for RIBs created by that entity. The network device administrator MAY allow reading of other RIBs by an external entity through access lists on the network device. The details of access lists are outside the scope of this document.

The data-model MUST support a full read of the RIB and subsequent incremental reads of changes to the RIB. An external agent SHOULD be able to request a full read at any time in the lifecycle of the connection. When sending data to an external entity, the RIB manager SHOULD try to send all dependencies of an object prior to sending that object.

4. Writing to the RIB

A RIB data-model MUST allow an external entity to write entries, for RIBs created by that entity. The network device administrator MAY allow writes to other RIBs by an external entity through access lists on the network device. The details of access lists are outside the scope of this document.

When writing an object to a RIB, the external entity SHOULD try to write all dependencies of the object prior to sending that object. The data-model SHOULD support requesting identifiers for nexthops and collecting the identifiers back in the response.

Route programming in the RIB MUST result in a return code that contains the following attributes:

- Installed - Yes/No (Indicates whether the route got installed in the FIB)
- Active - Yes/No (Indicates whether a route is fully resolved and is a candidate for selection)
- Reason - E.g. Not authorized

The data-model MUST specify which objects are modify-able objects. A modify-able object is one whose contents can be changed without having to change objects that depend on it and without affecting any data forwarding. To change a non-modifiable object, one will need to create a new object and delete the old one. For example, routes that use a nexthop that is identified by a nexthop identifier should be unaffected when the contents of that nexthop changes.

5. Notifications

Asynchronous notifications are sent by the network device’s RIB

manager to an external entity when some event occurs on the network device. A RIB data-model MUST support sending asynchronous notifications. A brief list of suggested notifications is as below:

- Route change notification, with return code as specified in Section 4
- Nexthop resolution status (resolved/unresolved) notification

6. RIB grammar

This section specifies the RIB information model in Routing Backus-Naur Form [RFC5511]. This grammar is intended to help the reader better understand the english text description in order to derive a data model. However it may not provide all the detail provided by the english text. When there is a lack of clarity in the grammar the english text will take precedence.

```
<routing-instance> ::= <INSTANCE_NAME>
                       [<interface-list>] <rib-list>
                       [<ROUTER_ID>]


<interface-list> ::= (<INTERFACE_IDENTIFIER> ...)


<rib-list> ::= (<rib> ...)
<rib> ::= <RIB_NAME> <rib-family>
                     [<route> ... ]
                     [ENABLE_IP_RPF_CHECK]
<rib-family> ::= <IPV4_RIB_FAMILY> | <IPV6_RIB_FAMILY> |
                 <MPLS_RIB_FAMILY> | <IEEE_MAC_RIB_FAMILY>


<route> ::= <match> <nexthop>
            [<route-attributes>]
            [<route-vendor-attributes>]


<match> ::= <IPV4> <ipv4-route> | <IPV6> <ipv6-route> |
            <MPLS> <MPLS_LABEL> | <IEEE_MAC> <MAC_ADDRESS> |
            <INTERFACE> <INTERFACE_IDENTIFIER>
<route-type> ::= <IPV4> | <IPV6> | <MPLS> | <IEEE_MAC> | <INTERFACE>


<ipv4-route> ::= <ip-route-type>
                 (<destination-ipv4-address> | <source-ipv4-address> |
                  (<destination-ipv4-address> <source-ipv4-address>))
<destination-ipv4-address> ::= <ipv4-prefix>
```

```
  <source-ipv4-address> ::= <ipv4-prefix>
  <ipv4-prefix> ::= <IPV4_ADDRESS> <IPV4_PREFIX_LENGTH>


  <ipv6-route> ::= <ip-route-type>
                   (<destination-ipv6-address> | <source-ipv6-address> |
                    (<destination-ipv6-address> <source-ipv6-address>))
  <destination-ipv6-address> ::= <ipv6-prefix>
  <source-ipv6-address> ::= <ipv6-prefix>
  <ipv6-prefix> ::= <IPV6_ADDRESS> <IPV6_PREFIX_LENGTH>
  <ip-route-type> ::= <SRC> | <DEST> | <DEST_SRC>


  <route-attributes> ::= <ROUTE_PREFERENCE> [<LOCAL_ONLY>]
                         [<address-family-route-attributes>]

  <address-family-route-attributes> ::= <ip-route-attributes> |
                                        <mpls-route-attributes> |
                                        <ethernet-route-attributes>
  <ip-route-attributes> ::= <>
  <mpls-route-attributes> ::= <>
  <ethernet-route-attributes> ::= <>
  <route-vendor-attributes> ::= <>


  <nexthop> ::= <nexthop-base> |
                (<NEXTHOP_LOAD_BALANCE> <nexthop-lb>) |
                (<NEXTHOP_PROTECTION> <nexthop-protection>) |
                (<NEXTHOP_REPLICATE> <nexthop-replicate>) |
                <nexthop-chain>

  <nexthop-base> ::= <NEXTHOP_ID> |
                     <nexthop-special> |
                     <EGRESS_INTERFACE> |
                     <ipv4-address> | <ipv6-address> |
                     (<EGRESS_INTERFACE>
                         (<ipv4-address> | <ipv6-address>)) |
                     (<EGRESS_INTERFACE> <IEEE_MAC_ADDRESS>) |
                     <tunnel-encap> | <tunnel-decap> |
                     <logical-tunnel> |
                     <RIB_NAME>)

  <EGRESS_INTERFACE> ::= <INTERFACE_IDENTIFIER>

  <nexthop-special> ::= <DISCARD> | <DISCARD_WITH_ERROR> |
                        (<RECEIVE> [<COS_VALUE>])
```

```
  <nexthop-lb> ::= <NEXTHOP_LB_WEIGHT> <nexthop>
                   (<NEXTHOP_LB_WEIGHT> <nexthop) ...


  <nexthop-protection> = <NEXTHOP_PREFERENCE> <nexthop>
                         (<NEXTHOP_PREFERENCE> <nexthop>)...

  <nexthop-replicate> ::= <nexthop> <nexthop> ...


  <nexthop-chain> ::= <nexthop> ...


  <logical-tunnel> ::= <tunnel-type> <TUNNEL_NAME>
  <tunnel-type> ::= <IPV4> | <IPV6> | <MPLS> | <GRE> | <VxLAN> | <NVGRE>

  <tunnel-encap> ::= (<IPV4> <ipv4-header>) |
                     (<IPV6> <ipv6-header>) |
                     (<MPLS> <mpls-header>) |
                     (<GRE> <gre-header>) |
                     (<VXLAN> <vxlan-header>) |
                     (<NVGRE> <nvgre-header>)

  <ipv4-header> ::= <SOURCE_IPv4_ADDRESS> <DESTINATION_IPv4_ADDRESS>
                    <PROTOCOL> [<TTL>] [<DSCP>]

  <ipv6-header> ::= <SOURCE_IPV6_ADDRESS> <DESTINATION_IPV6_ADDRESS>
                    <NEXT_HEADER> [<TRAFFIC_CLASS>]
                    [<FLOW_LABEL>] [<HOP_LIMIT>]

  <mpls-header> ::= (<mpls-label-operation> ...)
  <mpls-label-operation> ::= (<MPLS_PUSH> <MPLS_LABEL> [<S_BIT>]
                                          [<TOS_VALUE>] [<TTL_VALUE>]) |
                             (<MPLS_SWAP> <IN_LABEL> <OUT_LABEL>
                                          [<TTL_ACTION>])

  <gre-header> ::= <GRE_IP_DESTINATION> <GRE_PROTOCOL_TYPE> [<GRE_KEY>]
  <vxlan-header> ::= (<ipv4-header> | <ipv6-header>)
                     [<VXLAN_IDENTIFIER>]
  <nvgre-header> ::= (<ipv4-header> | <ipv6-header>)
                     <VIRTUAL_SUBNET_ID>
                     [<FLOW_ID>]

  <tunnel-decap> ::= ((<IPV4> <IPV4_DECAP> [<TTL_ACTION>]) |
                      (<IPV6> <IPV6_DECAP> [<HOP_LIMIT_ACTION>]) |
                      (<MPLS> <MPLS_POP> [<TTL_ACTION>]))
```

Figure 5: RIB rBNF grammar

6.1. Nexthop grammar explained

A nexthop is used to specify the next network element to forward the traffic to. It is also used to specify how the traffic should be load-balanced, protected using preference or multicasted using replication. This is explicitly specified in the grammar. The nexthop has recursion built-in to address complex use-cases like the one defined in Section 7.2.6.

7. Using the RIB grammar

The RIB grammar is very generic and covers a variety of features. This section provides examples on using objects in the RIB grammar and examples to program certain use cases.

7.1. Using route preference

Using route preference a client can pre-install alternate paths in the network. For example, if OSPF has a route preference of 10, then another client can install a route with route preference of 20 to the same destination. The OSPF route will get precedence and will get installed in the FIB. When the OSPF route is withdrawn, the alternate path will get installed in the FIB.

Route preference can also be used to prevent denial of service attacks by installing routes with the best preference, which either drops the offending traffic or routes it to some monitoring/analysis station. Since the routes are installed with the best preference, they will supersede any route installed by any other protocol.

7.2. Using different nexthops types

The RIB grammar allows one to create a variety of nexthops. This section describes uses for certain types of nexthops.

7.2.1. Tunnel nexthops

A tunnel nexthop points to a tunnel of some kind. Traffic that goes over the tunnel gets encapsulated with the tunnel encap. Tunnel nexthops are useful for abstracting out details of the network, by having the traffic seamlessly route between network edges. At the end of a tunnel, the tunnel will get decapsulated. Thus the grammar supports two kinds of operations, one for encap and another for decap.

7.2.2. Replication lists

One can create a replication list for replicating traffic to multiple destinations. The destinations, in turn, could be complex nexthops in themselves - at a level supported by the network device. Point to multipoint and broadcast are examples that involve replication.

A replication list (at the simplest level) can be represented as:

```
<nexthop> ::= <NEXTHOP_REPLICATE> <nexthop> [ <nexthop> ... ]
```

The above can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-replicate>
<nexthop> ::= <NEXTHOP_REPLICATE> <nexthop> <nexthop> ...
```

7.2.3. Weighted lists

A weighted list is used to load-balance traffic among a set of nexthops. From a modeling perspective, a weighted list is very similar to a replication list, with the difference that each member nexthop MUST have a NEXTHOP_LB_WEIGHT associated with it.

A weighted list (at the simplest level) can be represented as:

```
<nexthop> ::= <NEXTHOP_LOAD_BALANCE> (<nexthop> <NEXTHOP_LB_WEIGHT>)
                  [(<nexthop> <NEXTHOP_LB_WEIGHT>)... ]
```

The above can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-lb>
<nexthop> ::= <NEXTHOP_LOAD_BALANCE>
                <NEXTHOP_LB_WEIGHT> <nexthop>
                (<NEXTHOP_LB_WEIGHT> <nexthop>) ...
<nexthop> ::= <NEXTHOP_LOAD_BALANCE> (<NEXTHOP_LB_WEIGHT> <nexthop>)
                (<NEXTHOP_LB_WEIGHT> <nexthop>) ...
```

7.2.4. Protection

A primary/backup protection can be represented as:

```
<nexthop> ::= <NEXTHOP_PROTECTION> <1> <interface-primary>
                                   <2> <interface-backup>)
```

The above can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-protection>
<nexthop> ::= <NEXTHOP_PROTECTION> (<NEXTHOP_PREFERENCE> <nexthop>
                    (<NEXTHOP_PREFERENCE> <nexthop>)...)
<nexthop> ::= <NEXTHOP_PROTECTION> (<NEXTHOP_PREFERENCE> <nexthop>
                    (<NEXTHOP_PREFERENCE> <nexthop>))
<nexthop> ::= <NEXTHOP_PROTECTION> ((<NEXTHOP_PREFERENCE> <nexthop-base>
                    (<NEXTHOP_PREFERENCE> <nexthop-base>))
<nexthop> ::= <NEXTHOP_PROTECTION> (<1> <interface-primary>
                    (<2> <interface-backup>))
```

Traffic can be load-balanced among multiple primary nexthops and a single backup. In such a case, the nexthop will look like:

```
<nexthop> ::= <NEXTHOP_PROTECTION> (<1>
               (<NEXTHOP_LOAD_BALANCE>
                (<NEXTHOP_LB_WEIGHT> <nexthop-base>
                (<NEXTHOP_LB_WEIGHT> <nexthop-base>) ...))
                 <2> <nexthop-base>)
```

A backup can also have another backup. In such a case, the list will look like:

```
<nexthop> ::= <NEXTHOP_PROTECTION> (<1> <nexthop>
               <2> <NEXTHOP_PROTECTION>(<1> <nexthop> <2> <nexthop>))
```

7.2.5. Nexthop chains

A nexthop chain is a way to perform multiple operations on a packet by logically combining them. For example, when a VPN packet comes on the WAN interface and has to be forwarded to the correct VPN interface, one needs to POP the VPN label before sending the packet

out. Using a nexthop chain, one can chain together "pop MPLS header" and "send it out a specific EGRESS_INTERFACE".

The above example can be derived from the grammar as follows:

```
<nexthop-chain> ::= <nexthop> <nexthop>
<nexthop-chain> ::= <nexthop-base> <nexthop-base>
<nexthop-chain> ::= <tunnel-decap> <EGRESS_INTERFACE>
<nexthop-chain> ::= (<MPLS> <MPLS_POP>) <interface-outgoing>
```

Elements in a nexthop-chain are evaluated left to right.

A nexthop chain can also be used to put one or more headers on an outgoing packet. One example is a Pseudowire - which is MPLS over some transport (MPLS or GRE for instance). Another example is VxLAN over IP. A nexthop chain thus allows an external entity to break up the programming of the nexthop into independent pieces - one per encapsulation.

A simple example of MPLS over GRE can be represented as:

```
<nexthop-chain> ::= (<MPLS> <mpls-header>) (<GRE> <gre-header>)
                    <interface-outgoing>
```

The above can be derived from the grammar as follows:

```
<nexthop-chain> ::= <nexthop> <nexthop> <nexthop>
<nexthop-chain> ::= <nexthop-base> <nexthop-base> <nexthop-base>
<nexthop-chain> ::= <tunnel-encap> <tunnel-encap> <EGRESS_INTERFACE>
<nexthop-chain> ::= (<MPLS> <mpls-header>) (<GRE> <gre-header>)
                    <interface-outgoing>
```

7.2.6. Lists of lists

Lists of lists is a complex construct. One example of usage of such a construct is to replicate traffic to multiple destinations, with load balancing. In other words, for each branch of the replication tree, there are multiple interfaces on which traffic needs to be load-balanced on. So the outer list is a replication list for multicast and the inner lists are weighted lists for load balancing. Lets take an example of a network element has to replicate traffic to two other network elements. Traffic to the first network element should be load balanced equally over two interfaces outgoing-1-1 and

outgoing-1-2. Traffic to the second network element should be load balanced over three interfaces outgoing-2-1, outgoing-2-2 and outgoing-2-3 in the ratio 20:20:60.

This can be derived from the grammar as follows:

```
<nexthop> ::= <nexthop-replicate>
<nexthop> ::= <NEXTHOP_REPLICATE> (<nexthop> <nexthop>...)
<nexthop> ::= <NEXTHOP_REPLICATE> (<nexthop> <nexthop>)
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE> <nexthop-lb>)
              (<NEXTHOP_LOAD_BALANCE> <nexthop-lb>))
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE>
              (<NEXTHOP_LB_WEIGHT> <nexthop>
              (<NEXTHOP_LB_WEIGHT> <nexthop>) ...))
               ((<NEXTHOP_LOAD_BALANCE>
                (<NEXTHOP_LB_WEIGHT> <nexthop>
                (<NEXTHOP_LB_WEIGHT> <nexthop>) ...))
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE>
              (<NEXTHOP_LB_WEIGHT> <nexthop>
               (<NEXTHOP_LB_WEIGHT> <nexthop>)))
                ((<NEXTHOP_LOAD_BALANCE>
                (<NEXTHOP_LB_WEIGHT> <nexthop>
                (<NEXTHOP_LB_WEIGHT> <nexthop>)
                (<NEXTHOP_LB_WEIGHT> <nexthop>)))
<nexthop> ::= <NEXTHOP_REPLICATE> ((<NEXTHOP_LOAD_BALANCE>
               (<NEXTHOP_LB_WEIGHT> <nexthop>)
               (<NEXTHOP_LB_WEIGHT> <nexthop>)))
               ((<NEXTHOP_LOAD_BALANCE>
               (<NEXTHOP_LB_WEIGHT> <nexthop>)
               (<NEXTHOP_LB_WEIGHT> <nexthop>)
               (<NEXTHOP_LB_WEIGHT> <nexthop>)))
<nexthop> ::= <NEXTHOP_REPLICATE>
               ((<NEXTHOP_LOAD_BALANCE>
                 (50 <outgoing-1-1>)
                 (50 <outgoing-1-2>)))
                ((<NEXTHOP_LOAD_BALANCE>
                  (20 <outgoing-2-1>)
                  (20 <outgoing-2-2>)
                  (60 <outgoing-2-3>)))
```

7.3. Performing multicast

IP multicast involves matching a packet on (S, G) or (*, G), where both S (source) and G (group) are IP prefixes. Following the match, the packet is replicated to one or more recipients. How the recipients subscribe to the multicast group is outside the scope of this document.

In PIM-based multicast, the packets are IP forwarded on an IP multicast tree. The downstream nodes on each point in the multicast tree is one or more IP addresses. These can be represented as a replication list (Section 7.2.2).

In MPLS-based multicast, the packets are forwarded on a point to multipoint (P2MP) label-switched path (LSP). The nexthop for a P2MP LSP can be represented in the nexthop grammar as a <logical-tunnel> (P2MP LSP identifier) or a replication list (Section 7.2.2) of <tunnel-encap>, with each tunnel encap representing a single mpls downstream nexthop.

8. RIB operations at scale

This section discusses the scale requirements for a RIB data-model. The RIB data-model should be able to handle large scale of operations, to enable deployment of RIB applications in large networks.

8.1. RIB reads

Bulking (grouping of multiple objects in a single message) MUST be supported when a network device sends RIB data to an external entity. Similarly the data model MUST enable a RIB client to request data in bulk from a network device.

8.2. RIB writes

Bulking (grouping of multiple write operations in a single message) MUST be supported when an external entity wants to write to the RIB. The response from the network device MUST include a return-code for each write operation in the bulk message.

8.3. RIB events and notifications

There can be cases where a single network event results in multiple events and/or notifications from the network device to an external entity. On the other hand, due to timing of multiple things happening at the same time, a network device might have to send multiple events and/or notifications to an external entity. The network device originated event/notification message MUST support bulking of multiple events and notifications in a single message.

9. Security Considerations

All interactions between a RIB manager and an external entity MUST be

authenticated and authorized. The RIB manager MUST protect itself against a denial of service attack by a rogue external entity, by throttling request processing. A RIB manager MUST enforce limits on how much data can be programmed by an external entity and return error when such a limit is reached.

The RIB manager MUST expose a data-model that it implements. An external agent MUST send requests to the RIB manager that comply with the supported data-model. The data-model MUST specify the behavior of the RIB manager on handling of unsupported data requests.

10. IANA Considerations

This document does not generate any considerations for IANA.

11. Acknowledgements

The authors would like to thank Ron Folkes, Jeffrey Zhang, the working group co-chairs and reviewers on their comments and suggestions on this draft. The following people contributed to the design of the RIB model as part of the I2RS Interim meeting in April 2013 - Wes George, Chris Liljenstolpe, Jeff Tantsura, Susan Hares and Fabian Schneider.

12. References

12.1. Normative References

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

12.2. Informative References

[I-D.hares-i2rs-use-case-vn-vc]
Hares, S. and M. Chen, "Use Cases for Virtual Connections on Demand (VCoD) and Virtual Network on Demand (VNoD) using Interface to Routing System", draft-hares-i2rs-use-case-vn-vc-03 (work in progress), July 2014.

[I-D.white-i2rs-use-case]
White, R., Hares, S., and A. Retana, "Protocol Independent Use Cases for an Interface to the Routing System",

draft-white-i2rs-use-case-06 (work in progress), July 2014.

[RFC4915] Psenak, P., Mirtorabi, S., Roy, A., Nguyen, L., and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF", RFC 4915, DOI 10.17487/RFC4915, June 2007, <http://www.rfc-editor.org/info/rfc4915>.

[RFC5120] Przygienda, T., Shen, N., and N. Sheth, "M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs)", RFC 5120, DOI 10.17487/RFC5120, February 2008, <http://www.rfc-editor.org/info/rfc5120>.

[RFC5511] Farrel, A., "Routing Backus-Naur Form (RBNF): A Syntax Used to Form Encoding Rules in Various Routing Protocol Specifications", RFC 5511, DOI 10.17487/RFC5511, April 2009, <http://www.rfc-editor.org/info/rfc5511>.

[RFC7920] Atlas, A., Ed., Nadeau, T., Ed., and D. Ward, "Problem Statement for the Interface to the Routing System", RFC 7920, DOI 10.17487/RFC7920, June 2016, <http://www.rfc-editor.org/info/rfc7920>.

Authors' Addresses

Nitin Bahadur (editor)
Bracket Computing
150 West Evelyn Ave, Suite 200
Mountain View, CA 94041
US

Email: nitin_bahadur@yahoo.com

Sriganesh Kini (editor)

Email: sriganeshkini@gmail.com

Jan Medved
Cisco

Email: jmedved@cisco.com

I2RS WG
Internet-Draft
Intended status: Informational
Expires: October 6, 2016

D. Migault, Ed.
J. Halpern
Ericsson
S. Hares
Huawei
April 4, 2016

I2RS Environment Security Requirements
draft-ietf-i2rs-security-environment-reqs-01

Abstract

This document provides environment security requirements for the I2RS architecture. Environment security requirements are independent of the protocol used for I2RS. As a result, the requirements provided in this document are intended to provide good security practise so I2RS can be securely deployed and operated.

These security requirements are designated as environment security requirements as opposed to the protocol security requirements. The reason to have separate document is that protocol security requirements are intended to help the design of the I2RS protocol whether the environment requirements are rather intended for deployment or implementations.

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 October 6, 2016.

Table of Contents

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

This document provides environment security requirements for the I2RS architecture. Environment security requirements are independent of the protocol used for I2RS. As a result, the requirements provided in this document are intended to provide good security practise so I2RS can be securely deployed and operated.

These security requirements are designated as environment security requirements as opposed to the protocol security requirements described in [I-D.ietf-i2rs-protocol-security-requirements]. The reason to have separate document is that protocol security requirements are intended to help the design of the I2RS protocol whether the environment requirements are rather intended for deployment or implementations.

Even though I2RS is mostly concerned by the interface between the I2RS Client and the I2RS Agent, the security recommendations must consider the entire I2RS architecture, specifying where security functions may be hosted, and what should be met so to address any new attack vectors exposed by deploying this architecture. In other words, security has to be considered globally over the complete I2RS architecture and not only on the interfaces.

I2RS architecture depicted in [I-D.ietf-i2rs-architecture] describes the I2RS components and their interactions to provide a programmatic interface for the routing system. I2RS components as well as their interactions have not yet been considered in conventional routing systems. As such it introduces a need to interface with the routing system designated as I2RS plane in this document.

This document is built as follows. Section 4 describes how the I2RS plane can be contained or isolated from existing management plane, control plane and forwarding plane. The remaining sections of the document focuses on the security within the I2RS plane. Section 5 analyzes how the I2RS Access Control policies can be deployed throughout the I2RS plane in order to only grant access to the routing system resources to authorized components with the authorized privileges. This also includes providing a robust communication system between the components. Then, Section 6 details how I2RS keeps applications isolated one from another and do not affect the I2RS components. Applications may be independent, with different scopes, owned by different tenants. In addition, they modify the routing system that may be in an automatic way.

The reader is expected to be familiar with the [I-D.ietf-i2rs-architecture]. The document provides a list of

environment security requirements. Motivations are placed before the requirements are announced.

3. Terminology and Acronyms

- Environment Security Requirements :

- I2RS plane : The environment the I2RS process is running on. It includes the Applications, the I2RS Client and the I2RS Agent.

- I2RS user : The user of the I2RS client software or system.

- I2RS Access Control policies: policies controlling access of the routing resources by Applications. These policies are divided into policies applied by the I2RS Client regarding Applications and policies applied by the I2RS Agent regarding I2RS Clients.

- I2RS Client Access Control policies : The Access Control policies processed by the I2RS Client.

- I2RS Agent Access Control policies : The Access Control policies processed by the I2RS Agent.

4. I2RS Plane Isolation

Isolating the I2RS plane from other network plane, such as the control plane, is foundational to the security of the I2RS environment. Clearly differentiating I2RS components from the rest of the network protects the I2RS components from vulnerabilities in other parts of the network, and protect other systems vital to the health of the network from vulnerabilities in the I2RS plane. Separating the I2RS plane from other network control and forwarding planes is similar to the best common practice of containerizing software into modules, and defense in depth in the larger world of network security.

That said the I2RS plane cannot be considered as completely isolated from other planes, and interactions should be identified and controlled. Follows a brief description on how the I2RS plane positions itself in regard to the other planes. The description is indicative, and may not be exhaustive.

4.1. I2RS plane and management plane

The I2RS plane purpose is to provide a standard programmatic interface of the routing system resources to network oriented applications. Control plane and forwarding planes are related to routing protocols, and I2RS is based on top of those. The management

plane is usually vendor specific, provides a broader control over the networking equipment such as system service. Given its associated privileges it is expected to be reserved to highly trusted users like network administrators.

The I2RS plane and the management plane both interact with several common elements on forwarding and packet processing devices. [I-D.ietf-i2rs-architecture] describes several of these interaction points such as the local configuration, the static system state, routing, and signalling. Because of this potential overlaps, a routing resource may be accessed by different means (APIs, applications) and different planes. To keep these overlaps under control, one could either control the access to these resources with northbound APIs for example. Northbound APIs are provided to limit the scope of the applications toward the routing resources. In our case, the northbound API may be provided for the I2RS applications by the I2RS Client as well as to the management plane. In case conflicting overlaps cannot be avoided, and routing resource can be accessed by both the management plane and the I2RS plane, then, they should be resolved in a deterministic way.

On the northbound side, there must be clear protections against the I2RS system "infecting" the management system with bad information, or the management system "infecting" the I2RS system with bad information. The primary protection in this space is going to need to be validation rules on the speed of information flow, value limits on the data presented, and other protections of this type.

On the conflicting side/issues, there should be clear rules about which plan’s commands win in the case of conflict in order to prevent attacks where the two systems can be forced to deadlock.

4.2. I2RS plane and forwarding plane

Applications hosted on I2RS Client belongs to the I2RS plane. These Applications are hard to remain constrained into the I2RS plane, or even to limit their scope within the I2RS plane.

Applications using I2RS are part of the I2RS plane but may also interact with other components outside the I2RS plane. A common example may be an application uses I2RS to configure the network according to security or monitored events. As these events are monitored on the forwarding plane and not the I2RS plane, the application breaks plane isolation.

In addition, applications may communicate with multiple I2RS Clients; as such, any given application may have a broader view of the current and potential states of the network and the I2RS plane itself.

Because of this, any individual application could be an effective attack vector against the operation of the network, the I2RS plane, or any plane with which the I2RS plane interacts. There is little the I2RS plane can do to validate applications with which it interacts, other than to provide some broad general validations against common misconfigurations or errors. As with the separation between the management plane and the I2RS plane, this should minimally take the form of limits on information accepted, limits on the rate at which information is accepted, and rudimentary checks against intentionally formed routing loops or injecting information that would cause the control plane to fail to converge. Other forms of protection may be necessary.

4.3. I2RS plane and Control plane

The network control plane consists of the processes and protocols that discover topology, advertise reachability, and determine the shortest path between any location on the network and any destination. It is not anticipated there will be any interactions between the on-the-wire signalling used by the control plane. However, in some situations the I2RS system could modify information in the local databases of the control plane. This is not normally recommended, as it can bypass the normal loop free, loop free alternate, and convergence properties of the control plane. However, if the I2RS system does directly inject information into these tables, the I2RS system should ensure that loop free routing is preserved, including loop free alternates, tunnelled interfaces, virtual overlays, and other such constructions. Any information injected into the control plane directly could cause the control plane to fail to converge, resulting in a complete network outage.

4.4. Recommendations

To isolate I2RS transactions from other planes, it is recommended that:

REQ 1: Application-to-routing system resources communications should use an isolated communication channel. Various level of isolation can be considered. The highest level of isolation may be provided by using a physically isolated network. Alternatives may also consider logical isolation; for example by using vLAN. Eventually, in virtual environment that shares a common infrastructure, encryption, for example by using TLS or IPsec, may also be used as a way to enforce isolation.

REQ 2: The interface (like the IP address) used by the routing element to receive I2RS transactions should be a dedicated

physical or logical interface. As previously, mentioned a dedicated physical interface may contribute to a higher isolation, however logical isolation be also be considered for example by using a dedicated IP address or a dedicated port.

When the I2RS Agent performs an action on a routing element, the action is performed via process(es) associated to a system user . In a typical UNIX system, the user is designated with a user id (uid) and belong to groups designated by group ids (gid). These users are dependent of the routing element's operation system and are designated I2RS System Users. Some implementation may use a I2RS System User for the I2RS Agent that proxies the different I2RS Client, other implementations may use I2RS System User for each different I2RS Clients.

REQ 3: I2RS Agent should have permissions separate from any other entity (for example any internal system management processes or CLI processes).

I2RS resource may be shared with the management plane and the control plane. It is hardly possible to prevent interactions between the planes. I2RS routing system resource management is limited to the I2RS plane. As such, update of I2RS routing system outside of the I2RS plane may be remain unnoticed unless explicitly notified to the I2RS plane. Such notification is expected to trigger synchronization of the I2RS resource state within each I2RS component. This guarantees that I2RS resource are maintained in a coherent state among the I2RS plane. In addition, depending on the I2RS resource that is updated as well as the origin of the modification performed, the I2RS Access Control policies may be impacted. More especially, a I2RS Client is more likely to update an I2RS resources that has been updated by itself, then by the management plane for example.

REQ 4: I2RS plane should be informed when a routing system resource is modified by a user outside the I2RS plane access. The notification is not expected to flood the I2RS plane. Instead, notification is expected to be provided to the I2RS components interacting, configuring or monitoring the routing system resource. The notification is at least provided by the I2RS Agent to the various I2RS Client, but additional mechanisms might eventually be required so I2RS Client can relay the notification to the I2RS applications. This is designated as "I2RS resource modified out of I2RS plane". This requirements is also described in section 7.6 of [I-D.ietf-i2rs-architecture] for the I2RS Client. This document extends the requirement to the I2RS plane, in case future evolution of the I2RS plane.

REQ 5: I2RS plane should define an "I2RS plane overwrite policy". Such policy defines how an I2RS is able to update and overwrite a resource set by a user outside the I2RS plane. Such hierarchy has been described in section 6.3 and 7.8 of [I-D.ietf-i2rs-architecture]

5. I2RS Access Control for routing system resources

This section provides recommendations on how I2RS Access Control policies associated to the routing system resources. These policies only apply within the I2RS plane. More especially, the policies are associated to the Applications, the I2RS Clients and the I2RS Agents, with their associated identity and roles.

Note that the deployment of Applications, I2RS Client and I2RS Agent in a closed environment, should not be considered by default as a secure environment. Even for closed environment access control policies should be carefully defined to be able to, in the future to carefully extend the I2RS plane to remote Applications or remote I2RS Clients. As a result, this section always consider the case Applications and I2RS Client can be located locally, in a closed environment or distributed over open networks.

Although [I-D.ietf-i2rs-protocol-security-requirements] provides security requirements of the transport and protocol between the I2RS Client and the I2RS Agent, this section is mostly focused on access control.

5.1. I2RS Access Control architecture

Applications access to routing system resource via numerous intermediaries nodes. The application communicates with an I2RS Client. In some cases, the I2RS Client is only associated to a single application, but the I2RS Client may also act as a broker. The I2RS Client, then, communicates with the I2RS Agent that may eventually access the resource.

The I2RS Client broker approach provides scalability to the I2RS architecture as it avoids that each Application be registered to the I2RS Agent. Similarly, the I2RS Access Control should be able to scale numerous applications.

REQ 6: I2RS Access Control should be performed through the whole I2RS plane. It should not be enforced by the I2RS Agent only within the routing element. Instead, the I2RS Client should enforce the I2RS Client Access Control against Applications and the I2RS Agent should enforce the I2RS Agent Access Control against the I2RS Clients. Note that I2RS Client

Access Control is not in the scope of the I2RS architecture [I-D.ietf-i2rs-architecture], which exclusively focuses on the I2RS Agent Access Control.

This results in a layered and hierarchical or multi-party I2RS Access Control. An application will be able to access a routing system resource only if both the I2RS Client is granted access by the I2RS Agent and the application is granted access by the I2RS Client.

REQ 7: When an access request to a routing resource is refused by one party (the I2RS Client or the I2RS Agent), the initiator of the request (e.g the Application) as well as all intermediaries should indicate the reason the access has not been granted as well as the entity that has rejected the request.

REQ 8: In order to provide coherent Access Control policies enforced by multiple parties (e.g. the I2RS Client or the I2RS Agent), theses parties should trust each others, and communication between them should also be trusted, - that is should not introduce additional vector of attacks.

In case the I2RS Client Access Control or the I2RS Agent Access Control does not grant access to a routing system resource, the Application should be able to determine whether its request has been rejected by the I2RS Client or the I2RS Agent as well as the reason that caused the reject. More specifically, the I2RS Agent may reject the request because, for example, the I2RS Client is not an authorized I2RS Client, or because the I2RS Client does not not have enough privileges. The I2RS Client should be notified of the reason that caused the reject by the I2RS Agent, and The I2RS Client should return a message to the Application, indicating the I2RS Client is not authorized or does not have enough privileges. Similarly, if the I2RS Client does not grant the access to the Application, the I2RS Client should also inform the Application. The error message returned should be for example: "Read failure: you do not have the read permission", "Write failure: you do not have write permission" or "Write failure: resource accessed by someone else". This requirement has been written in a generic manner as it concerns various interactions: interactions between the application and the I2RS Client, interactions between the I2RS Client and the I2RS Agent. In the latest case, the requirement is part of the protocol security requirements addressed by [I-D.ietf-i2rs-protocol-security-requirements].

Although [I-D.ietf-i2rs-protocol-security-requirements] is focused on transport security requirements between the I2RS Client and the I2RS

Agent, the similar requirements may apply between the Application and the I2RS Client for a remote Application.

REQ 9: I2RS Client or I2RS Agent SHOULD also be able to refuse a communication with an Application or an I2RS Client when the communication channel does not fulfill enough security requirements. For example, the it should be able to reject messages over a communication channel that can be easily hijacked, like a clear text UDP channel.

In order to limit the number of access request that result in an error, each Application or I2RS Client may be able to retrieve the I2RS Access Control policies that applies to it. This subset of rules is designated as the "Individual I2RS Access Control policies". As these policies are subject to changes, a dynamic synchronization mechanism should be provided. However, such mechanism may be implemented with different level of completeness and dynamicity of the Individual I2RS Access Control policies. Caching requests that have been rejected may be one such variant. It remains relatively easy to implement and may avoid the complete disclosure of the Access Control policies of the I2RS Agent. In fact the relative disclosure of Access Control policies may leak confidential information in case of misconfiguration and should be balanced with the level of trust of the I2RS Client and the necessity of distributing the enforcement of the Access Control policies.

REQ 10: The I2RS Client may be able to request for its I2RS Access Control subset policies to the I2RS Agent or cache requests that have been rejected by the I2RS Agent to limit forwarding unnecessary queries to the I2RS Agent.

REQ 11: The I2RS Client may be able to be notified when its I2RS Access Control subset policies have been updated by the I2RS Agent.

Similarly, for the Applications

REQ 12: The Applications may be able to request for its I2RS Access Control subset policies, so to limit forwarding unnecessary queries to the I2RS Client.

REQ 13: The Applications may be able to subscribe a service that provides notification when its I2RS Access Control subset policies have been updated.

I2RS Access Control should be appropriately be balanced between the I2RS Client and the I2RS Agent. I2RS Access Control should not

solely rely only on the I2RS Client or the I2RS Agent as illustrated below:

- 1) I2RS Clients are dedicated to a single Application: In this case, it is likely that I2RS Access Control is enforced only by the I2RS Agent, as the I2RS Client is likely to accept all access request of the application. However, it is recommended that even in this case, I2RS Client Access Control is not based on an "Allow anything from application" policy, but instead the I2RS Client specifies accesses that are enabled. In addition, the I2RS Client may sync its associated I2RS Access Control policies with the I2RS Agent to limit the number of refused access requests being sent to the I2RS Agent. The I2RS Client is expected to balance pro and cons between sync its access control policies with the I2RS Agent and simply guessing the access request to the I2RS Agent.

- 2) A single I2RS Client acts as a broker for all Applications: In the case the I2RS Agent has a single I2RS Client. Such architecture results in I2RS Client with high privileges, as it sums the privileges of all applications. As end-to-end authentication is not provided between the Application and the I2RS Agent, if the I2RS Client becomes corrupted, it is possible for the malicious application escalates its privileges and make the I2RS Client perform some action on behalf of the application with more privileges. This would not have been possible with end-to-end authentication. In order to mitigate such attack, the I2RS Client that acts as a broker is expected to host application with an equivalent level of privileges.

REQ 14: The I2RS Access Control should explicitly specify accesses that are granted. More specifically, anything not explicitly granted -- the default rule-- should be denied.

In addition to distribute the I2RS Access Control policies between I2RS Clients and I2RS Agents, I2RS Access Control policies can also be distributed within a set of I2RS Clients or a set of I2RS Agents.

REQ 15: I2RS Clients should be distributed and act as brokers for Applications that share roughly similar permissions. This avoids ending with over privileges I2RS Client compared to hosted applications and thus discourages applications to perform privilege escalation within an I2RS Client.

REQ 16: I2RS Agents should be avoided being granted over privileges regarding to their authorized I2RS Client. I2RS Agent should be shared by I2RS Client with roughly similar permissions. More explicitly, an I2RS Agent shared between I2RS Clients

that are only provided read access to the routing system resources does not need to perform any write access, and so should not be provided these accesses. Suppose an I2RS Client requires write access to the resources. It is not recommended to grant the I2RS Agent the write access in order to satisfy a unique I2RS Client. Instead, the I2RS Client that requires write access should be connected to a I2RS Agent that is already shared by I2RS Client that requires a write access.

Access Control policies enforcement should be monitored in order to detect violation of the policies or detect an attack. Access Control policies enforcement may not be performed by the I2RS Client or the I2RS Agent as violation may require a more global view of the I2RS Access Control policies. As a result, consistency check and mitigation may instead be performed by the management plane. However, I2RS Clients and I2RS Agents play a central role.

REQ 17: I2RS Client and I2RS Agent should be able to log the various transaction they perform, as well as suspicious activities. These logs should be collected regularly and analyzed by functions that may be out of the I2RS plane.

Access Control policies should be implemented so that they remain manageable in short and longer term. This means the way they are managed today should be address future deployment and use of I2RS.

REQ 18: Access Control should be managed in an automated way, that is granting or revoking an Application should not involve manual configuration over the I2RS plane - like all the I2RS Clients.

REQ 19: Access Control should be scalable when the number of Application grows as well as when the number of I2RS Client increases. A typical implementation of a local I2RS Client Access Control policies may result in creating manually a system user associated to each Application. Such an approach is likely not to scale when the number of Applications increases or the number of I2RS Client increases.

REQ 20: Access Control should be dynamically managed and easy to be updated. Although the number of I2RS Clients is expected to be lower than the number of Application, as I2RS Agent provide access to the routing resource, it is of primary importance that an access can be granted or revoke in an efficient way.

REQ 21: I2RS Clients and I2RS Agents should be uniquely identified in the network to enable centralized management of the I2RS Access Control policies.

5.2. I2RS Agent Access Control policies

The I2RS Agent Access Control restricts the routing system resource access to authorized identities - possible access policies may be none, read or write. The initiator of an access request to a routing resource is always an Application. However, it remains challenging for the I2RS Agent to establish its access control policies based on the application that initiates the request. First, when an I2RS Client acts as a broker, the I2RS Agent may not be able to authenticate the Application. In that sense, the I2RS Agent relies on the capability of the I2RS Client to authenticate the Applications and apply the appropriated I2RS Client Access Control. Then, an I2RS Agent may not uniquely identify a piece of software implementing an I2RS Client. In fact, an I2RS Client may be provided multiple identities which can be associated to different roles or privileges. The I2RS Client is left responsible for using them appropriately according to the Application. Finally, each I2RS Client may contact various I2RS Agent with different privileges and Access Control policies.

This section provides recommendations on the I2RS Agent Access Control policies to keep I2RS Access Control coherent within the I2RS plane.

REQ 22: I2RS Agent Access Control policies should be primarily based on the I2RS Clients as described in [I-D.ietf-i2rs-architecture].

REQ 23: I2RS Agent Access Control policies may be based on the Application. In this case the identity of the Application MUST be authenticated by the I2RS Agent, and the secondary identity used to tag the application as defined in [I-D.ietf-i2rs-architecture] should be considered cautiously. The tag may be used associated only to an authenticated I2RS Client that is known to authenticate its Application.

The I2RS Agent Access Control policies may evolve over time as resource may also be updated outside the I2RS plane. Similarly, a given resource may be accessed by multiple I2RS users within the I2RS plane. Although this is considered as an error, depending on the I2RS Client that performed the update, the I2RS may accept or refuse to overwrite the routing system resource.

REQ 24: The I2RS Agent should know which identity (most likely system user) performed the latest update of the routing resource. This is true for an identity inside and outside the I2RS plane, so the I2RS Agent can appropriately perform an update according to the priorities associated to the requesting identity and the identity that last updated the resource. On an environment perspective, the I2RS Agent MUST be aware when the resource has been modified outside the I2RS plane, as well as its priority associated towards the I2RS plane. Similar requirements exist for identities within the I2RS plane, but belongs to the protocol security requirements.

REQ 25: the I2RS Agent should have a "I2RS Agent overwrite Policy" that indicates how identities can be prioritized. This requirements is also described in section 7.6 of [I-D.ietf-i2rs-architecture]. Similar requirements exist for components within the I2RS plane, but belongs to the protocol security requirements.

5.3. I2RS Client Access Control policies

The I2RS Client Access Control policies are responsible for authenticating the application managing the privileges for the applications, and enforcing access control to resources by the applications. As a result,

REQ 26: I2RS Client should authenticate its applications. If the I2RS Client acts as a broker and supports multiple Applications, it should authenticate each of them. Authentication of the application may used GSSAPI, Secure RPC mechanisms.

REQ 27: I2RS Client should define Access Control policies associated to each applications. An access to a routing resource by an Application should not be forwarded by the I2RS Client based on the I2RS Agent Access Control policies. The I2RS Client should first check whether the Application has sufficient privileges, and if so send an access request to the I2RS Agent. When an I2RS Client has multiple identities that are associated with different privileges. The I2RS Client Access Control policies should specify the associated I2RS Client’s identities, especially, when the I2RS Agent Access Control policies are changed for a given I2RS Client’s identity.

In case, no authentication mechanisms have being provided between the I2RS Client and the application, then I2RS Client may not act as broker, and be instead dedicated to a single application. By doing so, application authentication may rely on the I2RS authentication

mechanisms between the I2RS Client and the I2RS Agent. On the other hand, although this is not recommended, the I2RS Access Control policies is only enforced by the I2RS Agent.

5.4. Application and Access Control policies

Application does not enforce access control policies. Instead these are enforced by the I2RS Clients and the I2RS Agents. This section provides recommendations for Applications in order to ease I2RS Access Control by the I2RS Client and the I2RS Agent.

As multiple ways may be used for an Application to communicate with its associated I2RS Client, it is not expected that all Applications use the same conventional identifier format across the I2RS plane. However, if all Applications are running on a dedicated system sharing an I2RS Client, it is expected each Application may uniquely identified, for example using different system users.

REQ 28: Applications SHOULD be uniquely identified by their associated I2RS Clients

The I2RS Client provides access to resource on its behalf and this access should only be granted for trusted applications, or Applications with an similar level of trust. On the other hand, this does not prevent an I2RS Client to host a large number of Applications. Similarly, an Application may also require to access multiple I2RS Clients depending on the resource to be accessed. As I2RS Client are restricted for a subset of Applications,

REQ 29: Each Application SHOULD be associated to a restricted number of I2RS Client

REQ 30: Application SHOULD be provided means and methods to contact their associated I2RS Client. If the I2RS Client belongs to the Application (as a module or a library for example), or when the Application runs into a dedicated system (like a container) with a I2RS Client, it is obvious which I2RS Client the Application is associated to. On the other hand, Applications may also remotely access the I2RS Client. In this case, the Application is expected to be provided some means to be able to retrieve the necessary information to contact its associated I2RS Client. The IP address may not be appropriated in case renumbering occurs within the network or in case the traffic from Applications should be shared between multiple instances of a given I2RS Client. In this case a FQDN may be preferred.

6. I2RS Application Isolation

A key aspect of the I2RS architecture is the network oriented application. As these application are supposed to be independent, controlled by independent and various tenants. In addition to independent logic, these applications may be malicious. Then, these applications introduce also programmability which results in fast network settings.

The I2RS architecture should remain robust to these applications and make sure an application does not impact the other applications. This section discusses both security aspects related to programmability as well as application isolation in the I2RS architecture.

6.1. Robustness toward programmability

I2RS provides a programmatic interface in and out of the Internet routing system. This feature, in addition to the global network view provided by the centralized architecture comes with a few advantages in term of security.

The use of automation reduces configuration errors. In addition, this interface enables fast network reconfiguration. Agility provides a key advantage in term of deployment as side effect configuration may be easily addressed. Finally, it also provides facilities to monitor and mitigate an attack when the network is under attack.

On the other hand programmability also comes with a few drawbacks. First, applications can belong to multiple tenants with different objectives. This absence of coordination may result in unstable routing configurations such as oscillations between network configurations, and creation of loops for example. A typical example would be an application monitoring a state and changing its state. If another application performs the reverse operation, the routing system may become unstable. Data and application isolation is expected to prevent such situations to happen, however, to guarantee the network stability, constant monitoring and error detection are recommended to be activated.

REQ 31: The I2RS Agents should monitor constantly parts of the system for which I2RS Clients or Applications have provided requests. It should also be able to detect I2RS Clients or Applications that lead the routing system in an unstable state. Monitoring consists at least in logging events and eventually provide notifications or alerts to the management plane in case, something has been detected. The management

plane is in charge of collecting the logs, the notifications and eventually to consider the appropriated actions. A typical action may be the update of I2RS Access Control policies for example or re-configuring routing elements.

6.2. Application Isolation

6.2.1. DoS

Requirements for robustness to Dos Attacks have been addressed in the Communication channel section [I-D.ietf-i2rs-architecture].

The I2RS interface is used by application to interact with the routing states. As the I2RS Agent is shared between multiple applications, one application can prevent an application by performing DoS or DDoS attacks on the I2RS Agent or on the network. DoS attack targeting the I2RS Agent would consist in providing requests that keep the I2RS Agent busy for a long time. This may involve heavy computation by the I2RS Agent for example to blocking operations like disk access. In addition, DoS attacks targeting the network may use specific commands like monitoring stream over the network. Then, DoS attack may be also targeting the application directly by performing reflection attacks. Such an attack could be performed by indicating the target application as the target for some information like the listing of the RIB. Reflection may be performed at various levels and can be based on the use of UDP or at the service level like redirection of information to a specific repository.

REQ 32: In order to prevent DoS, it is recommended the I2RS Agent controls the resources allocated to each I2RS Clients. I2RS Client that acts as broker may not be protected as efficiently against these attacks unless they perform resource controls themselves of their hosted applications.

REQ 33: I2RS Agent does not make response redirection possible unless the redirection is previously validated and agreed by the destination.

REQ 34: avoid the use of underlying protocols that are not robust to reflection attacks.

6.2.2. Application Control

Requirements for Application Control have been addressed in the I2RS plane isolation as well as in the trusted Communication Channel sections.

Applications use the I2RS interface in order to update the routing system. These updates may be driven by behavior on the forwarding plane or any external behaviors. In this case, correlating observation to the I2RS traffic may enable to derive the application logic. Once the application logic has been derived, a malicious application may generate traffic or any event in the network in order to activate the alternate application.

REQ 35: Application logic should remain opaque to external listeners. Application logic may be partly hidden by encrypting the communication between the I2RS Client and the I2RS Agent. Additional ways to obfuscate the communications may involve sending random messages of various sizes. Such strategies have to be balanced with network load. Note that I2RS Client broker are more likely to hide the application logic compared to I2RS Client associated to a single application.

7. Security Considerations

The whole document is about security.

8. Privacy Considerations

9. IANA Considerations

10. Acknowledgments

A number of people provided a significant amount of helping comments and reviews. Among them the authors would like to thank Russ White, Russ Housley, Thomas Nadeau, Juergen Schoenwaelder, Jeffrey Haas, Alia Atlas, Linda Dunbar

11. References

11.1. Normative References

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

11.2. Informative References

[I-D.ietf-i2rs-architecture] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-09 (work in progress), March 2015.

[I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-01 (work in progress), September 2015.

Authors' Addresses

Daniel Migault (editor)
Ericsson
8400 boulevard Decarie
Montreal, QC H4P 2N2
Canada

Phone: +1 514-452-2160
Email: daniel.migault@ericsson.com

Joel Halpern
Ericsson

Email: Joel.Halpern@ericsson.com

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

J. Dong
X. Wei
Huawei Technologies

July 08, 2016

A YANG Data Model for Layer-2 Network Topologies
draft-ietf-i2rs-yang-l2-network-topology-03

Abstract

This document defines a YANG data model for Layer 2 network topologies.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 January 9, 2017.

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-i2rs-yang-network-topo] defines the YANG [RFC6020] [RFC6991] [I-D.ietf-netmod-rfc6020bis] data models of the abstract (generic) network and network topology. Such models can be augmented with technology-specific details to build more specific topology models.

This document defines the YANG data model for Layer 2 network topologies by augmenting the generic network and network topology data models with L2 specific topology attributes.

2. Layer 2 Topology Model

The Layer 2 network topology model is designed to be generic and applicable to Layer 2 networks built with different L2 technologies. It can be used to describe both the physical and the logical (virtual) L2 network topologies.

The Layer 2 topology model applies the generic network and network topology models to Layer 2 network topologies, and augments the generic models with information specific in Layer 2 networks. The relationship between the Layer 2 topology model and the generic network and network topology model is shown in the figure below:

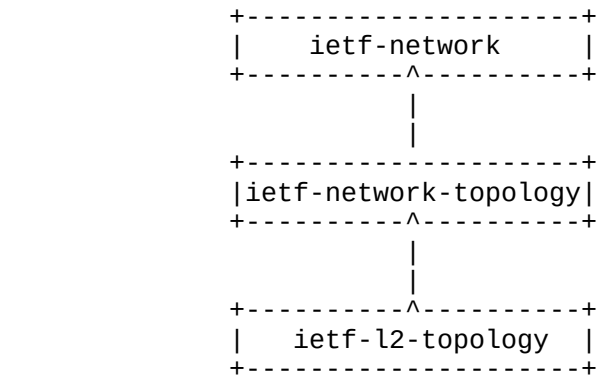


Figure 1. L2-topology model structure

In order to represent a Layer 2 network topology, the generic network and topology models are augmented with Layer-2 specific information, such as the identifiers, descriptions, attributes and states of the Layer-2 networks, nodes, links and termination points. Some of the information may be collected via Link Layer Discovery Protocol (LLDP) or other Layer-2 protocols, and some of them may be locally configured.

The structure of "ietf-l2-topology" data model is depicted in the following diagram. Brackets enclose list keys, "rw" means configuration data, "ro" means operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances.

```
module: ietf-l2-topology
augment /nw:networks/nw:network/nw:network-types:
   +--rw l2-network!
augment /nw:networks/nw:network:
   +--rw l2-network-attributes
      +--rw name?   string
      +--rw flag*   flag-type
augment /nw:networks/nw:network/nw:node:
   +--rw l2-node-attributes
      +--rw name?                 string
      +--rw description?          string
      +--rw management-address*   inet:ip-address
      +--rw sys-mac-address?      yang:mac-address
      +--rw management-vid?       vlan {VLAN}?
      +--rw nick-name*            trill-nickname {TRILL}?
      +--rw vn-id*                vni {VXLAN}?
      +--rw flag*                 flag-type
augment /nw:networks/nw:network/nt:link:
   +--rw l2-link-attributes
```

```
      +--rw name?    string
      +--rw flag*    flag-type
      +--rw rate?    decimal64
      +--rw delay?   uint32
      +--rw srlg*    uint32
augment /nw:networks/nw:network/nw:node/nt:termination-point:
   +--rw l2-termination-point-attributes
      +--rw description?          string
      +--rw maximum-frame-size?   uint32
      +--rw (l2-termination-point-type)?
      |  +--:(ethernet)
      |  |  +--rw mac-address?          yang:mac-address
      |  |  +--rw eth-encapsulation?    identityref
      |  |  +--rw port-vlan-id?         vlan {VLAN}?
      |  |  +--rw vlan-id-name* [vlan-id] {VLAN}?
      |  |     +--rw vlan-id      vlan
      |  |     +--rw vlan-name?   string
      |  +--:(legacy)
      |     +--rw layer-2-address?      yang:phys-address
      |     +--rw encapsulation?        identityref
      +--ro tp-state?             enumeration
notifications:
   +---n l2-node-event
   |  +--ro event-type?           l2-network-event-type
   |  +--ro network-ref?          -> /nw:networks/network/network-id
   |  +--ro node-ref?             -> /nw:networks/network[nw:network-id = curren
t()/../network-ref]/node/node-id
   |  +--ro l2-network!
   |  +--ro l2-node-attributes
   |     +--ro name?                  string
   |     +--ro description?           string
   |     +--ro management-address*    inet:ip-address
   |     +--ro sys-mac-address?       yang:mac-address
   |     +--ro management-vid?        vlan {VLAN}?
   |     +--ro nick-name*             trill-nickname {TRILL}?
   |     +--ro vn-id*                 vni {VXLAN}?
   |     +--ro flag*                  flag-type
   +---n l2-link-event
   |  +--ro event-type?           l2-network-event-type
   |  +--ro network-ref?          -> /nw:networks/network/network-id
   |  +--ro link-ref?             -> /nw:networks/network[nw:network-id = curren
t()/../network-ref]/nt:link/link-id
   |  +--ro l2-network!
   |  +--ro l2-link-attributes
   |     +--ro name?    string
   |     +--ro flag*    flag-type
   |     +--ro rate?    decimal64
   |     +--ro delay?   uint32
   |     +--ro srlg*    uint32
   +---n l2-termination-point-event
```

```
      +--ro event-type?                          l2-network-event-type
      +--ro network-ref?                         -> /nw:networks/network/network-i
d
      +--ro node-ref?                            -> /nw:networks/network[nw:networ
k-id = current()/../network-ref]/node/node-id
      +--ro tp-ref?                              -> /nw:networks/network[nw:networ
k-id = current()/../network-ref]/node[nw:node-id = current()/../node-ref]/nt:ter
mination-point/tp-id
      +--ro l2-network!
      +--ro l2-termination-point-attributes
         +--ro description?          string
         +--ro maximum-frame-size?   uint32
         +--ro (l2-termination-point-type)?
         |  +--:(ethernet)
         |  |  +--ro mac-address?          yang:mac-address
         |  |  +--ro eth-encapsulation?    identityref
         |  |  +--ro port-vlan-id?         vlan {VLAN}?
         |  |  +--ro vlan-id-name* [vlan-id] {VLAN}?
         |  |     +--ro vlan-id      vlan
         |  |     +--ro vlan-name?   string
         |  +--:(legacy)
         |     +--ro layer-2-address?      yang:phys-address
         |     +--ro encapsulation?        identityref
         +--ro tp-state?             enumeration
```

The L2-topology module augments the generic ietf-network and ietf-network-topology modules as follows:

- A new network type "l2-network-type" is introduced. This is represented by a container object, and is inserted under the "network-types" container of the generic ietf-network module in [I-D.ietf-i2rs-yang-network-topo].

- Additional network attributes are introduced in a grouping "l2-network-attributes", which augments the "network" list of the ietf-network module. The attributes include Layer-2 network name and a set of flags. Each type of flag is represented by a separate identity.

- Additional data objects for Layer-2 nodes are introduced by augmenting the "node" list of the generic ietf-network module. New objects include Layer-2 node identifier, description, management address, and a set of flags.

- Additional data objects for Layer-2 termination points are introduced by augmenting the "termination-point" list of the ietf-network-topology module defined in [I-D.ietf-i2rs-yang-network-topo]. New objects include Layer-2 termination point descriptions, Layer-2 termination point type specific attributes and Layer-2 termination point states.

- Links in the ietf-network-topology module are augmented as well with a set of Layer-2 parameters, allowing to associate a link with a name, a set of Layer-2 link attributes and flags.

- The optional L2 technology specific attributes are introduced in this module as Layer-2 features.

3. Layer-2 Topology Yang Module

```
<CODE BEGINS> file "ietf-l2-topology@2016-07-07.yang"
module ietf-l2-topology {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-l2-topology";
    prefix "l2t";

    import ietf-network {
      prefix "nw";
    }

    import ietf-network-topology {
      prefix "nt";
    }

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

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

    organization
      "IETF I2RS (Interface to the Routing System) Working Group";
    contact
      "WG Web:   <http://tools.ietf.org/wg/i2rs/>
       WG List:  <mailto:i2rs@ietf.org>
       WG Chair:  Susan Hares
                 <mailto:shares@ndzh.com>

       WG Chair:  Russ White
                 <mailto:russ@riw.us>

       Editor:    Jie Dong
                 <mailto:jie.dong@huawei.com>

       Editor:    Xiugang Wei
                 <mailto:weixiugang@huawei.com>";
```

```
       description
         "This module defines a basic model for
          the layer-2 topology of a network";

       revision "2016-07-07" {
         description "Initial revision";
         reference "draft-ietf-i2rs-l2-network-topology-03";
       }

      /*
       * Typedefs
       */

       typedef vlan {
         type uint16 {
           range "0..4095";
         }
         description "VLAN ID";
       }

       typedef trill-nickname {
         type uint16;
         description "TRILL Nickname";
       }

       typedef vni {
         type uint32 {
           range "1..16777215";
         }
         description "VxLAN Network Identifier";
       }

       typedef flag-type {
         type identityref {
           base "flag-identity";
         }
         description "Base type for flags";
       }

       typedef l2-network-event-type {
         type enumeration {
           enum "add" {
             value 0;
             description "An L2 node or link or termination-point
             has been added";
           }
           enum "remove" {
             value 1;
```

```
            description "An L2 node or link or termination-point
            has been removed";
          }
          enum "update" {
            value 2;
            description "An L2 node or link or termination-point
            has been updated";
          }
        }
        description "l2 network event type for notifications";
      } // l2-topology-event-type



      /*
      * Features
      */

      feature VLAN {
        description
          "Indicates that the system supports the
           vlan functions";
      }

      feature QinQ {
        description
          "Indicates that the system supports the
           qinq functions";
      }

      feature PBB {
        description
         "Indicates that the device supports the
          provider-backbone-bridging functions";
      }

      feature VPLS {
        description
          "Indicates that the device supports the
           VPLS functions";
        reference "RFC 4761, RFC 4762";
      }

      feature TRILL {
        description
          "Indicates that the device supports the
           TRILL functions";
        reference "RFC 6325";
```

```
       }

       feature VXLAN {
         description
           "Indicates that the device supports the
            VXLAN functions";
         reference "RFC 7348";
       }

      /*
       * Identities
       */


       identity flag-identity {
         description "Base type for flags";
       }

       identity encapsulation-type {
         description
           "Base identity from which specific encapsulation
            types are derived.";
       }

       identity eth-encapsulation-type {
         base encapsulation-type;
         description
           "Base identity from which specific ethernet
            encapsulation types are derived.";

       }

       identity ethernet {
         base eth-encapsulation-type;
         description
           "native ethernet encapsulation";
       }

       identity vlan {
         base eth-encapsulation-type;
         description
           "vlan encapsulation";
       }

       identity qinq {
         base eth-encapsulation-type;
         description
           "qinq encapsulation";
```

```
  }

  identity pbb {
    base eth-encapsulation-type;
    description
      "pbb encapsulation";
  }

  identity trill {
    base eth-encapsulation-type;
    description
      "trill encapsulation";
  }

  identity vpls {
    base eth-encapsulation-type;
    description
      "vpls encapsulation";
  }

  identity vxlan {
    base eth-encapsulation-type;
    description
      "vxlan encapsulation";
  }

  identity frame-relay {
    base encapsulation-type;
    description
      "Frame Relay encapsulation";
  }

  identity ppp {
    base encapsulation-type;
    description
      "PPP encapsulation";
  }

  identity hdlc {
    base encapsulation-type;
    description
      "HDLC encapsulation";
  }

  identity atm {
    base encapsulation-type;
    description
      "Base identity from which specific ATM
```

```
        encapsulation types are derived.";

    }

    identity pwe3 {
      base encapsulation-type;
      description
        "Base identity from which specific pw
         encapsulation types are derived.";
    }


   /*
    * Groupings
    */


    grouping l2-network-type {
      description "Identify the topology type to be L2.";
      container l2-network {
        presence "indicates L2 Network";
        description
        "The presence of the container node indicates
         L2 Topology";
      }
    }

    grouping l2-network-attributes {
      description "L2 Topology scope attributes";
      container l2-network-attributes {
        description "Containing L2 network attributes";
        leaf name {
          type string;
          description "Name of the L2 network";
        }

        leaf-list flag {
          type flag-type;
          description "L2 network flags";
        }
      }
    }

    grouping l2-node-attributes {
      description "L2 node attributes";
      container l2-node-attributes {
        description "Containing L2 node attributes";
        leaf name {
```

```
            type string;
            description "Node name";
          }
          leaf description {
            type string;
            description "Node description";
          }
          leaf-list management-address {
            type inet:ip-address;
            description "System management address";
          }
          leaf sys-mac-address {
            type yang:mac-address;
            description "System MAC-address";
          }
          leaf management-vid {
            if-feature VLAN;
            type vlan;
            description "System management VID";
          }
          leaf-list nick-name {
            if-feature TRILL;
            type trill-nickname;
            description "Nickname of the RBridge";
          }
          leaf-list vn-id {
            if-feature VXLAN;
            type vni;
            description "VNI of the VxLAN";
          }
          leaf-list flag {
            type flag-type;
            description "Node operational flags";
          }
        }
      }  // grouping l2-node-attributes

      grouping l2-link-attributes {
        description "L2 link attributes";
        container l2-link-attributes {
          description "Containing L2 link attributes";
          leaf name {
            type string;
            description "Link name";
          }
          leaf-list flag {
            type flag-type;
            description "Link flags";
```

```
          }
          leaf rate {
            type decimal64 {
              fraction-digits 2;
            }
            description "Link rate";

          }
          leaf delay {
            type uint32;
            description "Link delay in microseconds";
          }
          leaf-list srlg {
            type uint32;
            description
              "List of Shared Risk Link Groups
               this link belongs to.";
          }
        }
      } // grouping l2-link-attributes

      grouping l2-termination-point-attributes {
        description "L2 termination point attributes";
        container l2-termination-point-attributes {
          description "Containing L2 TP attributes";
          leaf description {
            type string;
            description "Port description";
          }

          leaf maximum-frame-size {
            type uint32;
            description "Maximum frame size";
          }

          choice l2-termination-point-type {
            description
              "Indicates termination-point type
               specific attributes";
            case ethernet {
              leaf mac-address {
                type yang:mac-address;
                description "Interface MAC address";
              }

              leaf eth-encapsulation {
                type identityref {
                  base eth-encapsulation-type;
```

```
            }
            description
              "Encapsulation type of this
               ternimation point.";
          }

          leaf port-vlan-id {
            if-feature VLAN;
            type vlan;
            description "Port VLAN ID";
          }

          list vlan-id-name {
            if-feature VLAN;
            key "vlan-id";
            description "Interface configured VLANs";
            leaf vlan-id {
              type vlan;
              description "VLAN ID";
            }
            leaf vlan-name {
              type string;
              description "VLAN Name";
            }
          }
        } //case ethernet

        case legacy {
          leaf layer-2-address {
            type yang:phys-address;
            description "Interface Layer 2 address";
          }

          leaf encapsulation {
            type identityref {
              base encapsulation-type;
            }
            description
              "Encapsulation type of this termination point.";
          }
        } //case legacy

      } //choice termination-point-type

      leaf tp-state {
        type enumeration {
          enum in-use {
            value 0;
```

```
                description
                  "the termination point is in forwarding state";
              }
              enum blocking {
                value 1;
                description
                  "the termination point is in blocking state";
              }
              enum down {
                value 2;
                description
                  "the termination point is in down state";
              }
              enum others {
                value 3;
                description
                  "the termination point is in other state";
              }
            }
            config false;
            description "State of the termination point";
          }
        }
      } // grouping l2-termination-point-attributes

   /*** grouping of network/node/link/tp leaf-refs ***/

     grouping network-ref {
       description
         "Grouping for an absolute reference to a network topology
         instance.";
       leaf network-ref {
         type leafref {
           path "/nw:networks/nw:network/nw:network-id";
         }
         description
           "An absolute reference to a network topology instance.";
       }
     }

     grouping link-ref {
       description
         "Grouping for an absolute reference to a link instance.";
         uses network-ref;
         leaf link-ref {
           type leafref {
             path "/nw:networks/nw:network"
               +"[nw:network-id = current()/../network-ref]"
```

```
            +"/nt:link/nt:link-id";
        }
      description
        "An absolute reference to a link instance.";
    }
  }

  grouping node-ref {
    description
      "Grouping for an absolute reference to a node instance.";
    uses network-ref;
    leaf node-ref {
      type leafref {
        path "/nw:networks/nw:network"
          +"[nw:network-id = current()/../network-ref]"
          +"/nw:node/nw:node-id";
      }
      description
        "An absolute reference to a node instance.";
    }
  }

  grouping tp-ref {
    description
      "Grouping for an absolute reference to a termination point.";
    uses node-ref;
    leaf tp-ref {
      type leafref {
        path "/nw:networks/nw:network"
          +"[nw:network-id = current()/../network-ref]"
          +"/nw:node[nw:node-id = current()/../node-ref]"
          +"/nt:termination-point/nt:tp-id";
      }
      description
        "Grouping for an absolute reference to a TP.";
    }
  }


   /*
    * Data nodes
    */


    augment "/nw:networks/nw:network/nw:network-types" {
      description
        "Introduce new network type for L2 topology";
      uses l2-network-type;
```

```
      }

      augment "/nw:networks/nw:network" {
        when "/nw:networks/nw:network/nw:network-types/l2-network" {
          description
            "Augmentation parameters apply only for networks
             with L2 topology";
        }
        description
          "Configuration parameters for the L2 network
           as a whole";
        uses l2-network-attributes;
      }

      augment "/nw:networks/nw:network/nw:node" {
        when "/nw:networks/nw:network/nw:network-types/l2-network" {
          description
            "Augmentation parameters apply only for networks
             with L2 topology";
        }
        description
          "Configuration parameters for L2 at the node
           level";
        uses l2-node-attributes;
      }

      augment "/nw:networks/nw:network/nt:link" {
        when "/nw:networks/nw:network/nw:network-types/l2-network" {
          description
            "Augmentation parameters apply only for networks
             with L2 topology";
        }
        description "Augment L2 topology link information";
        uses l2-link-attributes;
      }

      augment "/nw:networks/nw:network/nw:node/nt:termination-point" {
        when "/nw:networks/nw:network/nw:network-types/l2-network" {
          description
            "Augmentation parameters apply only for networks
             with L2 topology";
        }
        description
          "Augment L2 topology termination point information";
        uses l2-termination-point-attributes;
      }

     /*
```

```
      * Notifications
      */

      notification l2-node-event {
        description "Notification event for L2 node";
        leaf event-type {
          type l2-network-event-type;
          description "Event type";
        }
        uses node-ref;
        uses l2-network-type;
        uses l2-node-attributes;
      }

      notification l2-link-event {
        description "Notification event for L2 link";
        leaf event-type {
          type l2-network-event-type;
          description "Event type";
        }
        uses link-ref;
        uses l2-network-type;
        uses l2-link-attributes;
      }

      notification l2-termination-point-event {
        description "Notification event for L2 termination point";
        leaf event-type {
          type l2-network-event-type;
          description "Event type";
        }
        uses tp-ref;
        uses l2-network-type;
        uses l2-termination-point-attributes;
      }

  }  // module l2-topology
  <CODE ENDS>
```

4. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

5. Security Considerations

The transport protocol used for sending the topology data MUST support authentication and SHOULD support encryption. The data-model by itself does not create any security implications.

6. Acknowledgements

The authors would like to acknowledge the comments and suggestions received from Susan Hares, Alia Atlas, Juergen Schoenwaelder, Mach Chen, Alexander Clemm and Sriganesh Kini.

7. References

7.1. Normative References

[I-D.ietf-i2rs-yang-network-topo]
Clemm, A., Medved, J., Varga, R., Tkacik, T., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A Data Model for Network Topologies", draft-ietf-i2rs-yang-network-topo-03 (work in progress), June 2016.

[I-D.ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", draft-ietf-netmod-rfc6020bis-14 (work in progress), June 2016.

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

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

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

7.2. Informative References

[I-D.amante-i2rs-topology-use-cases]
Medved, J., Previdi, S., Lopez, V., and S. Amante, "Topology API Use Cases", draft-amante-i2rs-topology-use-cases-01 (work in progress), October 2013.

[I-D.medved-i2rs-topology-requirements]
Medved, J., Previdi, S., Gredler, H., Nadeau, T., and S. Amante, "Topology API Requirements", draft-medved-i2rs-topology-requirements-00 (work in progress), February 2013.

Authors' Addresses

Jie Dong
Huawei Technologies
Huawei Campus, No. 156 Beiqing Rd.
Beijing 100095
China

Email: jie.dong@huawei.com

Xiugang Wei
Huawei Technologies
Huawei Campus, No. 156 Beiqing Rd.
Beijing 100095
China

Email: weixiugang@huawei.com

I2RS Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

Y. Zhuang
D. Shi
Huawei
R. Gu
China Mobile
H. Ananthakrishnan
Packet Design
July 3, 2017

A YANG Data Model for Fabric Topology in Data Center Network
draft-zhuang-i2rs-yang-dc-fabric-network-topology-04

Abstract

This document defines a YANG data model for fabric topology in Data Center Network.

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 January 4, 2018.

Copyright Notice

Table of Contents

1. Introduction

Normally, a data center network is composed of single or multiple fabrics which are also known as PODs (a Point Of Delivery). These fabrics may be heterogeneous due to implementation of different technologies while DC network upgrading or enrolling new techniques and features. For example, Fabric A may use VXLAN while Fabric B may use VLAN within a DC network. Likewise, a legacy Fabric may use VXLAN while a new Fabric B implemented technique discussed in NVO3 WG such as GPE[I-D. draft-ietf-nvo3-vxlan-gpe] may be built due to DC expansion and upgrading. The configuration and management of such DC networks with heterogeneous fabrics will be sophisticated and complex.

Luckily, for a DC network, a fabric can be considered as an atomic structure to provide network services and management, as well as expand network capacity. From this point of view, the miscellaneous DC network management can be decomposed to task of managing each fabric respectively along with their connections, which can make the entire management much concentrated and flexible, also easy to expand.

With this purpose, this document defines a YANG data model for the Fabric-based Data center topology by using YANG [6020][7950]. To do

so, it augments the generic network and network topology data models defined in [I-D.ietf-i2rs-yang-network-topo] with information specific to Data Center fabric network.

This model defines the generic configuration and operational state for a fabric-based network topology, which can be extended by vendors with specific information. This model can then be used by a network controller to represent its view of the fabric topology that it controls and expose it to network administrators or applications for DC network management.

With the context of topology architecture defined in [I-D.ietf-i2rs-yang-network-topo] and [I.D. draft-ietf-i2rs-usecase-reqs-summary], this model can also be treated as an application of I2RS network topology model [I-D.ietf-i2rs-yang-network-topo] in the scenario of Data center network management. It can also act as a service topology when mapping network elements at fabric layer to elements to other topologies, such as L3 topology defined in [I.D. draft-ietf-i2rs-yang-l3-topology-01.

By using this fabric topology model, people can treat a fabric as an entity and focus on characteristics of fabrics (such as encapsulation type, gateway type, etc.) as well as their interconnections while putting the underlay topology aside. As such, clients can consume the topology information at fabric level, while no need to be aware of entire set of links and nodes in underlay networks. The configuration of a fabric topology can be made by a network administractor to the controller by adding physical devices and links of a fabric into a fabric network. Alternatively, the fabric topology can also learnt from the underlay network infrastructure.

2. Definitions an Acronyms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2.1. Terminology

DC Fabric: also known as POD, is a module of network, compute, storage, and application components that work together to deliver networking services. It is a repeatable design pattern, and its components maximize the modularity, scalability, and manageability of data centers.

2.2. Tree diagram

The following notations are used within the data tree and carry the meaning as below.

Each node is printed as:
<status> <flags> <name> <opts> <type>

<status> is one of:
+ for current
x for deprecated
o for obsolete
<flags> is one of:
rw for configuration data
ro for non-configuration data
-x for rpcs
-n for notifications
<name> is the name of the node

If the node is augmented into the tree from another module, its name is printed as <prefix>:<name>.
<opts> is one of:
? for an optional leaf or choice
! for a presence container
* for a leaf-list or list
[<keys>] for a list's keys
<type> is the name of the type for leafs and leaf-lists

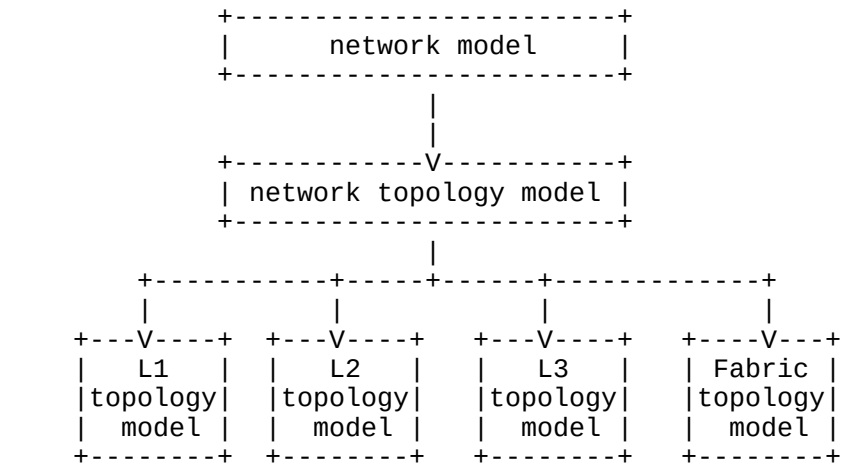
In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

3. Model Overview

This section provides an overview of the DC Fabric topology model and its relationship with other topology models.

3.1. Topology Model structure

The relationship of the DC fabric topology model and other topology models is shown in the following figure (dotted lines in the figure denote augmentations).



From the perspective of resource management and service provisioning for a Data Center network, the fabric topology model augments the basic network topology model with definitions and features specific to a DC fabric, to provide common configuration and operations for heterogeneous fabrics.

3.2. Fabric Topology Model

The fabric topology model module is designed to be generic and can be applied to data center fabrics built with different technologies, such as VLAN, VXLAN etc al. The main purpose of this module is to configure and manage fabrics and their connections. provide a fabric-based topology view for data center network applications.

3.2.1. Fabric Topology

In the fabric topology module, a fabric is modeled as a node in the network, while the fabric-based Data center network consists of a set of fabric nodes and their connections known as "fabric port". The following is the snatch of the definition to show the main structure of the model:

```
module: ietf-fabric-topology
augment /nw:networks/nw:network/nw:network-types:
   +--rw fabric-network!
augment /nw:networks/nw:network/nw:node:
   +--rw fabric-attribute
          +--rw name?           string
          +--rw type?           fabrictype:underlayer-network-type
          +--rw description?    string
          +--rw options
          +--...
augment /nw:networks/nw:network/nw:node/nt:termination-point:
   +--ro fport-attribute
          +--ro name?          string
          +--ro role?          fabric-port-role
          +--ro type?          fabric-port-type
```

The fabric topology module augments the generic ietf-network and ietf-network-topology modules as follows:

- A new topology type "ietf-fabric-topology" is introduced and added under the "network-types" container of the ietf-network module.

- Fabric is defined as a node under the network/node container. A new container of "fabric-attribute" is defined to carry attributes for a fabric network such as gateway mode, fabric types, involved device nodes and links etc al.

- Termination points (in network topology module) are augmented with fabric port attributes defined in a container. The "termination-point" here can represent the "port" of a fabric that provides connections to other nodes, such as device internally, another fabric externally and also end hosts.

Details of fabric node and fabric termination point extension will be explained in the following sections.

3.2.2. Fabric node extension

As a network, a fabric itself is composed of set of network elements i.e. devices, and related links. As stated previously, the configuration of a fabric is contained under the "fabric-attribute" container depicted as follows:

```
  +--rw fabric-attribute
     +--rw fabric-id?      fabric-id
     +--rw name?           string
     +--rw type?           fabrictype:underlayer-network-type
     +--rw vni-capacity
     |  +--rw min?   int32
     |  +--rw max?   int32
     +--rw description?    string
     +--rw options
     |  +--rw gateway-mode?           enumeration
     |  +--rw traffic-behavior?       enumeration
     |  +--rw capability-supported*   fabrictype:service-capabilities
     +--rw device-nodes* [device-ref]
     |  +--rw device-ref    fabrictype:node-ref
     |  +--rw role?         fabrictype:device-role
     +--rw device-links* [link-ref]
     |  +--rw link-ref    fabrictype:link-ref
     +--rw device-ports* [port-ref]
        +--rw port-ref     fabrictype:tp-ref
        +--rw port-type?   enumeration
        +--rw bandwith?    Enumeration
```

As in the module, additional data objects for nodes are introduced by augmenting the "node" list of the network module. New objects include fabric name, type of the fabric, descriptions of the fabric as well as a set of options defined in an "options" container. The options container includes type of the gateway-mode (centralized or distributed) and traffic-behavior (whether acl needed for the traffic).

Also, it defines a list of device-nodes and related links as supporting-nodes to form a fabric network. These device nodes and links are leaf-ref of existing nodes and links in the physical topology. For the device-node, the "role" object is defined to represents the role of the device within the fabric, such as "SPINE" or "LEAF", which should work together with gateway-mode.

3.2.3. Fabric termination-point extension

Since the fabric is considered as a node, in this concept, "termination-points" can represent "ports" of a fabric that connects to other fabrics or end hosts, besides representing ports that connect devices inside the fabric itself.

As such, the "termination-point" in the fabric topology has three roles, including internal TP that connects to devices within a

fabric, external TP that connects to outside network, as well as access TP to end hosts.

A set of "termination-point" indicates all connections of a fabric including its internal connections, interconnections with other fabrics and also connections to end hosts for a DC network.

The structure of fabric ports is as follows:

```
augment /nw:networks/nw:network/nw:node/nt:termination-point:
   +--ro fport-attribute
      +--ro name?          string
      +--ro role?          fabric-port-role
      +--ro type?          fabric-port-type
      +--ro device-port?   tp-ref
      +--ro (tunnel-option)?
         +--:(gre)
            +--ro src-ip?        inet:ip-prefix
            +--ro dest-ip?       inet:ip-address
```

It augments the termination points (in network topology module) with fabric port attributes defined in a container.

New nodes are defined for fabric ports which include name, role of the port within the fabric (internal port, external port to outside network, access port to end hosts), port type (l2 interface, l3 interface etc al). By using the device-port defined as a tp-ref, this fabric port can be mapped to a device node in the underlay network.

Also, a new container for tunnel-options is introduced as well to present the tunnel configuration on the port.

The terminiation points information are all learnt from the underlay networks but not configured by the fabric topology layer.

4. Fabric YANG Module

<CODE BEGINS> file "ietf-fabric-types@2016-09-29.yang"

```
module ietf-fabric-types {

    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-fabric-types";
    prefix fabrictypes;

    import ietf-inet-types { prefix "inet"; revision-date "2013-07-15"; }
```

```
        import ietf-network-topology { prefix nt;  }

        organization
    "IETF I2RS (Interface to the Routing System) Working Group";

        contact
    "WG Web:    <http://tools.ietf.org/wg/i2rs/ >
     WG List:   <mailto:i2rs@ietf.org>

     WG Chair:  Susan Hares
                <mailto:shares@ndzh.com>

     WG Chair:  Russ White
                <mailto:russ@riw.us>

     Editor:    Yan Zhuang
                <mailto:zhuangyan.zhuang@huawei.com>

     Editor:    Danian Shi
                <mailto:shidanian@huawei.com>";

    description
        "This module contains a collection of YANG definitions for Fabric.";

    revision "2016-09-29" {
        description
            "Initial revision of faas.";
                reference
                        "draft-zhuang-i2rs-yang-dc-fabric-network-topology-02";
    }

    identity fabric-type {
                description
                        "base type for fabric networks";
    }

    identity vxlan-fabric {
        base fabric-type;
                description
                        "vxlan fabric";
    }

    identity vlan-fabric {
        base fabric-type;
                description
                        "vlan fabric";
    }
```

```
    typedef service-capabilities {
        type enumeration {
            enum ip-mapping {
                                description "NAT";
                            }
            enum acl-redirect{
                                description "acl redirect, which can provide SFC
 function";
                            }
            enum dynamic-route-exchange{
                                description "dynamic route exchange";
                            }
        }
                description
                        "capability of the device";
    }

    /*
     * Typedefs
     */
    typedef node-ref {
        type instance-identifier;
        description "A reference to a node in topology";
    }

    typedef tp-ref {
        type instance-identifier;
        description "A reference to a termination point in topology";
    }

    typedef link-ref {
                type instance-identifier;
        description "A reference to a link in topology";
    }

    typedef device-role {
        type enumeration {
            enum SPINE {
                description "a spine node";
            }
            enum LEAF {
                description "a leaf node";
            }
            enum BORDER {
                description "a border node";
            }
        }
        default "LEAF";
        description "device role type";
```

```
    }

    typedef fabric-port-role {
        type enumeration {
            enum internal {
                description "the port used for devices to access each other.";
            }
            enum external {
                description "the port used for fabric to access outside network.
";
            }
            enum access {
                description "the port used for Endpoint to access fabric.";
            }
            enum reserved {
                description " not decided yet. ";
            }
        }
                description "the role of the physical port ";
    }

    typedef fabric-port-type {
        type enumeration {
            enum layer2interface {
                                description "l2 if";
            }
            enum layer3interface {
                                description "l3 if";
            }
            enum layer2Tunnel {
                                description "l2 tunnel";
            }
            enum layer3Tunnel {
                                description "l3 tunnel";
            }
        }
                description
                        "fabric port type";
    }

    typedef underlayer-network-type {
        type enumeration {
            enum VXLAN {
                description "vxlan";
            }
            enum TRILL {
                description "trill";
            }
            enum VLAN {
```

```
                description "vlan";
            }
        }
                description "";
    }

    typedef layer2-protocol-type-enum {
        type enumeration {
            enum VLAN{
                                description "vlan";
                        }
            enum VXLAN{
                                description "vxlan";
                        }
            enum TRILL{
                                description "trill";
                        }
            enum NvGRE{
                                description "nvgre";
                        }
        }
                description "";
    }

    typedef access-type {
        type enumeration {
            enum exclusive{
                                description "exclusive";
                        }
            enum vlan{
                                description "vlan";
                        }
        }
        description "";
    }

    grouping fabric-port {
                description
                        "attributes of a fabric port";
        leaf name {
            type string;
                        description "name of the port";
        }
        leaf role {
            type fabric-port-role;
                        description "role of the port in a fabric";
        }
        leaf type {
```

```
            type fabric-port-type;
                        description "type of the port";
        }
        leaf device-port {
            type tp-ref;
                        description "the device port it mapped to";
        }
        choice tunnel-option {
                        description "tunnel options";

            case gre {
                 leaf src-ip {
                    type inet:ip-prefix;
                                        description "source address";
                }
                leaf dest-ip {
                    type inet:ip-address;
                                        description "destination address";
                }
            }
        }
    }

    grouping route-group {
                description
                        "route attributes";
        list route {
            key "destination-prefix";
                        description "route list";

            leaf description {
                type string;
                description "Textual description of the route.";
            }
            leaf destination-prefix {
                type inet:ipv4-prefix;
                mandatory true;
                description "IPv4 destination prefix.";
            }
            choice next-hop-options {
                                description "choice of next hop options";
                case simple-next-hop {
                    leaf next-hop {
                        type inet:ipv4-address;
                        description "IPv4 address of the next hop.";
                    }
                    leaf outgoing-interface {
                        type nt:tp-id;
```

```
                        description "Name of the outgoing interface.";
                    }
                }
            }
        }
    }

    grouping port-functions {
                description
                        "port functions";

        container port-function {
                        description "port functions";
            choice function-type {
                                description "type of functions";
                case ip-mapping {
                    list ip-mapping-entry {
                                                key "external-ip";
                                                description "list of NAT entry";
                        leaf external-ip {
                            type inet:ipv4-address;
                                                        description "external ad
dress";
                        }
                        leaf internal-ip {
                            type inet:ipv4-address;
                                                        description "internal ad
dress";
                        }
                    }
                }
            }
        }
    }
    grouping acl-list {
                description "acl list";
        list fabric-acl {
                        key fabric-acl-name;
                        description "fabric acl list";
            leaf fabric-acl-name {
                type string;
                                description "acl name";
            }
        }
    }
}
<CODE ENDS>

<CODE BEGINS> file "ietf-fabric-topology@2017-03-10.yang"
module ietf-fabric-topology {
```

```
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-fabric-topology";
    prefix fabric;

        import ietf-network { prefix nw;  }
        import ietf-network-topology { prefix nt;  }
    import ietf-fabric-types { prefix fabrictype; revision-date "2016-09-29"; }

        organization
    "IETF I2RS (Interface to the Routing System) Working Group";

        contact
   "WG Web:    <http://tools.ietf.org/wg/i2rs/ >
    WG List:   <mailto:i2rs@ietf.org>

    WG Chair:  Susan Hares
               <mailto:shares@ndzh.com>

    WG Chair:  Russ White
               <mailto:russ@riw.us>

    Editor:    Yan Zhuang
               <mailto:zhuangyan.zhuang@huawei.com>

    Editor:    Danian Shi
               <mailto:shidanian@huawei.com>";


    description
        "This module contains a collection of YANG definitions for Fabric.

               Copyright (c) 2016 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
               (http://trustee.ietf.org/license-info).

               This version of this YANG module is part of
               draft-zhuang-i2rs-yang-dc-fabric-network-topology;
               see the RFC itself for full legal notices.";

    revision "2017-03-10" {
               description
                       "remove the rpcs and add extra attributes";
```

```
                reference
                        "draft-zhuang-i2rs-yang-dc-fabric-network-topology-03";
        }
    revision "2016-09-29" {
        description
            "Initial revision of fabric topology.";
                reference
                        "draft-zhuang-i2rs-yang-dc-fabric-network-topology-02";
    }

    identity fabric-context {
                description
                        "identity of fabric context";
    }

    typedef fabric-id {
        type nw:node-id;
        description
            "An identifier for a fabric in a topology.
            The identifier is generated by compose-fabric RPC.";
    }

        //grouping statements
        grouping fabric-network-type {
      description "Identify the topology type to be fabric.";
      container fabric-network {
        presence "indicates fabric Network";
        description
        "The presence of the container node indicates
         fabric Topology";
      }
    }

    grouping fabric-options {
                description "options for a fabric";

        leaf gateway-mode {
            type enumeration {
                enum centralized {
                    description "centerilized gateway";
                }
                enum distributed {
                    description "distributed gateway";
                }
            }
            default "distributed";
                        description "gateway mode";
        }
```

```
        leaf traffic-behavior {
            type enumeration {
                enum normal {
                    description "normal";
                }
                enum policy-driven {
                    description "policy driven";
                }
            }
            default "normal";
                        description "traffic behavior of the fabric";
        }

        leaf-list capability-supported {
            type fabrictype:service-capabilities;
                        description
                                "supported services of the fabric";
        }
    }

    grouping device-attributes {
                description "device attributes";
        leaf device-ref {
            type fabrictype:node-ref;
                        description
                                "the device it includes to";
        }
        leaf role {
            type fabrictype:device-role;
            default "LEAF";
                        description
                                "role of the node";
        }
    }

    grouping link-attributes {
                description "link attributes";
        leaf link-ref {
            type fabrictype:link-ref;
                        description
                                "the link it includes";
        }
    }

        grouping port-attributes {
                description "port attributes";
        leaf port-ref {
            type fabrictype:tp-ref;
```

```
                    description
                            "port reference";
        }
        leaf port-type {
            type enumeration {
                enum ETH {
                                    description "ETH";
                            }
                enum SERIAL {
                                    description "Serial";
                            }
            }
                    description
                            "port type: ethernet or serial";
        }
        leaf bandwith {
            type enumeration {
                enum 1G {
                                    description "1G";
                            }
                enum 10G {
                                    description "10G";
                            }
                enum 40G {
                                    description "40G";
                            }
                enum 100G {
                                    description "100G";
                            }
                enum 10M {
                                    description "10M";
                            }
                enum 100M {
                                    description "100M";
                            }
                enum 1M {
                                    description "1M";
                            }
            }
                    description
                            "bandwidth on the port";
        }
    }

    grouping fabric-attributes {
                description "attributes of a fabric";

                leaf fabric-id {
```

```
                type fabric-id;
                                description
                                        "fabric id";
        }

        leaf name {
            type string;
                        description
                                "name of the fabric";
        }

        leaf type {
            type fabrictype:underlayer-network-type;
            description
                                "The type of physical network that implements th
is fabric.Examples are vlan, and trill.";
        }

                container vni-capacity {
            description "number of vnis the fabric has";
            leaf min {
                type int32;
                                description
                                        "vni min capacity";
            }

            leaf max {
                type int32;
                                description
                                        "vni max capacity";
            }
        }

        leaf description {
            type string;
                        description
                                "description of the fabric";
        }

        container options {
                        description "options of the fabric";
            uses fabric-options;
        }

        list device-nodes {
                        key device-ref;
                        description "include device nodes in the fabric";
            uses device-attributes;
        }
```

```
        list device-links {
                        key link-ref;
                        description "include device links within the fabric";
            uses link-attributes;
        }

                list device-ports {
            key port-ref;
                        description "include device ports within the fabric";
            uses port-attributes;
        }

    }

        // augment statements

        augment "/nw:networks/nw:network/nw:network-types" {
    description
      "Introduce new network type for Fabric-based logical topology";

                uses fabric-network-type;
        }

    augment "/nw:networks/nw:network/nw:node" {
        when "/nw:networks/nw:network/nw:network-types/fabric-network" {
        description
          "Augmentation parameters apply only for networks
           with fabric topology";
      }
        description "Augmentation for fabric nodes created by faas.";

        container fabric-attribute {
                        description
                        "attributes for a fabric network";

            uses fabric-attributes;
        }
    }

    augment "/nw:networks/nw:network/nw:node/nt:termination-point" {
        when "/nw:networks/nw:network/nw:network-types/fabric-network" {
        description
          "Augmentation parameters apply only for networks
           with fabric topology";
      }
        description "Augmentation for port on fabric.";

        container fport-attribute {
```

```
            config false;
                        description
                        "attributes for fabric ports";
            uses fabrictype:fabric-port;
        }
    }
}
<CODE ENDS>
```

5. Security Consideration

TBD

6. Acknowledgements

7. References

7.1. Normative References

[I-D.draft-ietf-i2rs-yang-l3-topology]
Clemm, A., Medved, J., Tkacik, T., Liu, X., Bryskin, I., Guo, A., Ananthakrishnan, H., Bahadur, N., and V. Beeram, "A YANG Data Model for Layer 3 Topologies", I-D draft-ietf-i2rs-yang-l3-topology-04, September 2016.

[I-D.draft-ietf-i2rs-yang-network-topo]
Clemm, A., Medved, J., Tkacik, T., Varga, R., Bahadur, N., and H. Ananthakrishnan, "A YANG Data Model for Network Topologies", I-D draft-ietf-i2rs-yang-network-topo-06, September 2016.

[I-D.draft-ietf-nvo3-vxlan-gpe]
Maino, F., Kreeger, L., and U. Elzur, "Generic Protocol Extension for VXLAN", I-D draft-ietf-i2rs-yang-network-topo-02, October 2016.

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

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

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

[RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, Auguest 2016.

7.2. Informative References

[I-D.draft-ietf-i2rs-usecase-reqs-summary] Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", I-D draft-ietf-i2rs-usecase-reqs-summary-01, May 2015.

Appendix A. Non NMDA -state modules

```
        <CODE BEGINS> file "ietf-fabric-topology-state@2017-06-29.yang"
        module ietf-fabric-topology-state {

    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-fabric-topology-state";
    prefix sfabric;

        import ietf-network { prefix nw;  }
    import ietf-fabric-types { prefix fabrictype; revision-date "2016-09-29"; }
        import ietf-fabric-topology {prefix fp;}
        organization
    "IETF I2RS (Interface to the Routing System) Working Group";

        contact
        "WG Web:    <http://tools.ietf.org/wg/i2rs/ >
     WG List:   <mailto:i2rs@ietf.org>

     WG Chair:  Susan Hares
                <mailto:shares@ndzh.com>

     WG Chair:  Russ White
                <mailto:russ@riw.us>

     Editor:    Yan Zhuang
                <mailto:zhuangyan.zhuang@huawei.com>

     Editor:    Danian Shi
                <mailto:shidanian@huawei.com>";


    description
        "This module contains a collection of YANG definitions for Fabric topolo
gy state for non NMDA.

                Copyright (c) 2016 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
                (http://trustee.ietf.org/license-info).

                This version of this YANG module is part of
                draft-zhuang-i2rs-yang-dc-fabric-network-topology;
                see the RFC itself for full legal notices.";

    revision "2017-06-29"{
                description
                        "update to NMDA compliant format";
                reference
                        "draft-zhuang-i2rs-yang-dc-fabric-network-topology-04";
       }

       //grouping statements
       grouping fabric-network-type {
      description "Identify the topology type to be fabric.";
      container fabric-network {
        presence "indicates fabric Network";
        description
        "The presence of the container node indicates
         fabric Topology";
      }
    }

    grouping fabric-options {
                description "options for a fabric";

        leaf gateway-mode {
            type enumeration {
                enum centralized {
                    description "centerilized gateway";
                }
                enum distributed {
                    description "distributed gateway";
                }
            }
            default "distributed";
                        description "gateway mode";
        }

        leaf traffic-behavior {
            type enumeration {
                enum normal {
```

```
                    description "normal";
                }
                enum policy-driven {
                    description "policy driven";
                }
            }
            default "normal";
                        description "traffic behavior of the fabric";
        }

        leaf-list capability-supported {
            type fabrictype:service-capabilities;
                        description
                                "supported services of the fabric";
        }
    }

    grouping device-attributes {
                description "device attributes";
        leaf device-ref {
            type fabrictype:node-ref;
                        description
                                "the device it includes to";
        }
        leaf role {
            type fabrictype:device-role;
            default "LEAF";
                        description
                                "role of the node";
        }
    }

    grouping link-attributes {
                description "link attributes";
        leaf link-ref {
            type fabrictype:link-ref;
                        description
                                "the link it includes";
        }
    }

        grouping port-attributes {
                description "port attributes";
        leaf port-ref {
            type fabrictype:tp-ref;
                        description
                                "port reference";
        }
```

```
        leaf port-type {
            type enumeration {
                enum ETH {
                                        description "ETH";
                                }
                enum SERIAL {
                                        description "Serial";
                                }
            }
                        description
                                "port type: ethernet or serial";
        }
        leaf bandwith {
            type enumeration {
                enum 1G {
                                        description "1G";
                                }
                enum 10G {
                                        description "10G";
                                }
                enum 40G {
                                        description "40G";
                                }
                enum 100G {
                                        description "100G";
                                }
                enum 10M {
                                        description "10M";
                                }
                enum 100M {
                                        description "100M";
                                }
                enum 1M {
                                        description "1M";
                                }
            }
                        description
                                "bandwidth on the port";
        }
    }

    grouping fabric-attributes {
                description "attributes of a fabric";

                leaf fabric-id {
                type fp:fabric-id;
                                description
                                        "fabric id";
```

```
        }

        leaf name {
            type string;
                        description
                                "name of the fabric";
        }

        leaf type {
            type fabrictype:underlayer-network-type;
            description
                                "The type of physical network that implements th
is fabric.Examples are vlan, and trill.";
        }

                container vni-capacity {
            description "number of vnis the fabric has";
            leaf min {
                type int32;
                                description
                                        "vni min capacity";
            }

            leaf max {
                type int32;
                                description
                                        "vni max capacity";
            }
        }

        leaf description {
            type string;
                        description
                                "description of the fabric";
        }

        container options {
                        description "options of the fabric";
            uses fabric-options;
        }

        list device-nodes {
                        key device-ref;
                        description "include device nodes in the fabric";
            uses device-attributes;
        }

        list device-links {
                        key link-ref;
```

```
                        description "include device links within the fabric";
            uses link-attributes;
        }

                list device-ports {
            key port-ref;
                        description "include device ports within the fabric";
            uses port-attributes;
        }

    }

        // augment statements

        augment "/nw:networks/nw:network/nw:network-types" {
    description
      "Introduce new network type for Fabric-based logical topology";

                uses fabric-network-type;
        }

    augment "/nw:networks/nw:network/nw:node" {
        when "/nw:networks/nw:network/nw:network-types/fabric-network" {
        description
          "Augmentation parameters apply only for networks
           with fabric topology.";
      }
        description "Augmentation for fabric nodes.";

        container fabric-attribute-state {
                        config false;
                        description
                        "attributes for a fabric network";

            uses fabric-attributes;
        }
    }
}
```

<CODE ENDS>

Authors' Addresses

Yan Zhuang
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: zhuangyan.zhuang@huawei.com

Danian Shi
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: shidanian@huawei.com

Rong Gu
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing, Beijing 100053
China

Email: gurong_cmcc@outlook.com

Hariharan Ananthakrishnan
Packet Design

Email: hari@packetdesign.com