Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 15, 2018

L. Wang
Individual
M. Chen
Huawei
A. Dass
Ericsson
H. Ananthakrishnan
Packet Design
S. Kini
Individual
N. Bahadur
Bracket Computing
February 11, 2018

A YANG Data Model for Routing Information Base (RIB)
draft-ietf-i2rs-rib-data-model-10

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

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 https://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 August 15, 2018.

Table of Contents

1. Introduction

The Interface to the Routing System (I2RS) [RFC7921] 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

YANG tree diagrams provide a concise representation of a YANG module, and SHOULD be included to help readers understand YANG module structure. Guidelines on tree diagrams can be found in Section 3 of [I-D.ietf-netmod-yang-tree-diagrams].

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-definition
         +--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-definition
      |  |  +---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-definition
      +---n route-change
         +--ro rib-name                string
```

```
        +--ro address-family              rib-family-definition
        +--ro route-index              uint64
        +--ro match
        |  +--ro (route-type)?
        |     +--:(ipv4)
        |     |  ...
        |     +--:(ipv6)
        |     |  ...
        |     +--:(mpls-route)
        |     |  ...
        |     +--:(mac-route)
        |     |  ...
        |     +--:(interface-route)
        |        ...
        +--ro route-installed-state route-installed-state-definition
        +--ro route-state         route-state-definition
        +--ro route-change-reason route-reason-definition
```

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-definition
      +--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].
- ACTIVE: Indicates whether a route has at least one fully resolved nexthop and is therefore eligible for installation in the FIB.
- 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-encapsulation

      * tunnel-decapsulation

      * rib-name

   o Chain: Provide a way to perform multiple operations on a packet by logically combining them.

   o Load-balance: Designed for load-balance case where it normally will have multiple weighted nexthops.

   o Protection: Designed for protection scenario where it normally will have primary and standby nexthop.

   o 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-definition
   |     +--:(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-definition
```

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-definition
|        +--:(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-definition
|        |        +--:(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-decapsulation-nexthop) {nexthop-tunnel}?
|        |  +--rw tunnel-decapsulation
|        |     +--rw (tunnel-type)?
|        |        +--:(ipv4) {ipv4-tunnel}?
|        |        |  +--rw ipv4-decapsulation
|        |        |     +--rw ipv4-decapsulation tunnel-decapsulation-action-def
inition
|        |        |     +--rw ttl-action?   ttl-action-definition
|        |        +--:(ipv6) {ipv6-tunnel}?
|        |        |  +--rw ipv6-decapsulation
|        |        |     +--rw ipv6-decapsulation tunnel-decapsulation-action-def
inition
|        |        |     +--rw hop-limit-action? hop-limit-action-definition
|        |        +--:(mpls) {mpls-tunnel}?
|        |           +--rw label-pop
```

```
|       |             +--rw label-pop     mpls-label-action-definition
|       |             +--rw ttl-action?   ttl-action-definition
|       +--:(logical-tunnel-nexthop) {nexthop-tunnel}?
|       |  +--rw logical-tunnel
|       |     +--rw tunnel-type tunnel-type-definition
|       |     +--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 but during situations when the nexthop already exists and the nexthop-id is known, this action is not expected.. The output is a combination of the route operation states while querying the appropriate node in the data tree 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-definition
   |  |  +---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-definition
      +---n route-change
         +--ro rib-name                string
         +--ro address-family              rib-family-definition
         +--ro route-index             uint64
         +--ro match
         |  +--ro (route-type)?
         |     +--:(ipv4)
         |     |  ...
         |     +--:(ipv6)
         |     |  ...
         |     +--:(mpls-route)
         |     |  ...
         |     +--:(mac-route)
         |     |  ...
         |     +--:(interface-route)
         |        ...
         +--ro route-installed-state route-installed-state-definition
         +--ro route-state             route-state-definition
         +--ro route-change-reason     route-change-reason-definition
```

Figure 7: Notifications Structure

3. YANG Modules

<CODE BEGINS> file "ietf-i2rs-rib@2017-12-05.yang"

```
module ietf-i2rs-rib {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-i2rs-rib";
  // replace with iana namespace when assigned
  prefix "iir";

  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991";
  }

  import ietf-interfaces {
    prefix if;
        reference "RFC 7223";
  }

  import ietf-yang-types {
    prefix yang;
        reference "RFC 6991";
  }

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

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

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

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

     Editor:   Hariharan Ananthakrishnan
               <mailto:hari@packetdesign.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.
          Copyright (c) <2018> IETF Trust and the persons
      identified as authors of the code.  All rights reserved.";
   revision "2018-02-12" {
     description "initial revision";
     reference "draft-ietf-i2rs-data-model-10";


         // RFC Ed.: replace XXXX with actual RFC number and remove
         // this note


   }

   //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-definition {
     type identityref {
       base "mpls-label-action";
     }
     description
       "MPLS label action definition.";
   }

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

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

   identity ipv6-decapsulation {
```

```
    base "tunnel-decapsulation-action";
    description
      "IPv4 tunnel decapsulation.";
  }

  typedef tunnel-decapsulation-action-definition {
    type identityref {
      base "tunnel-decapsulation-action";
    }
    description
      "Tunnel decapsulation definition.";
  }

  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-definition {
     type identityref {
       base "ttl-action";
     }
     description
       "TTL action definition.";
   }

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

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

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

   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-definition {
     type identityref {
       base "rib-family";
     }
     description
       "RIB address family definition.";
   }

   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-definition {
     type identityref {
       base "route-type";
     }
     description
       "Route type definition.";
   }

   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-definition {
    type identityref {
      base "tunnel-type";
    }
    description
      "Tunnel type definition.";
  }

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

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

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

 //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-definition {
     type identityref {
       base "route-change-reason";
     }
     description
       "Route change reason definition.";
   }

   typedef nexthop-preference-definition {
     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-definition {
      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-definition;
        config false;
        description
          "Indicate a route's state: Active or Inactive.";
      }
      leaf route-installed-state {
        type route-installed-state-definition;
        config false;
        description
          "Indicate that a route's installed states:
           Installed or uninstalled.";
      }
      leaf route-reason {
        type route-change-reason-definition;
        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-definition;
        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-definition;
        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-definition;
          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-decapsulation-nexthop {
          if-feature nexthop-tunnel;
          container tunnel-decapsulation {
            uses tunnel-decapsulation;
            description
              "This is to specify the decapsulation of 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-definition;
       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-definition;
              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-decapsulation {
    description
      "Tunnel decapsulation information.";
    choice tunnel-type {
      description
        "Nexthop tunnel type options.";
      case ipv4 {
        if-feature ipv4-tunnel;
        container ipv4-decapsulation {
          description
            "IPv4 decapsulation.";
          leaf ipv4-decapsulation  {
            type tunnel-decapsulation-action-definition;
            mandatory true;
            description
              "IPv4 decapsulation operations.";
          }
          leaf ttl-action {
            type ttl-action-definition;
            description
              "The ttl actions:
               no-action or copy to inner header.";
          }
        }
      }
      case ipv6 {
        if-feature ipv6-tunnel;
        container ipv6-decapsulation {
          description
            "IPv6 decapsulation.";
          leaf ipv6-decapsulation {
```

```
            type tunnel-decapsulation-action-definition;
            mandatory true;
            description
              "IPv6 decapsulation operations.";
          }
          leaf hop-limit-action {
            type hop-limit-action-definition;
            description
              "The hop limit actions:
               no-action or copy to inner header.";
          }
        }
      }
      case mpls {
        if-feature mpls-tunnel;
        container label-pop {
          description
            "MPLS decapsulation.";
          leaf label-pop {
            type mpls-label-action-definition;
            mandatory true;
            description
              "Pop a label from the label stack.";
          }
          leaf ttl-action {
            type ttl-action-definition;
            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-definition;
        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-definition;
         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-definition;
      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-definition;
      mandatory true;
      description
        "The address family of a RIB.";
    }
    uses route-prefix;
    leaf route-installed-state {
      type route-installed-state-definition;
      mandatory true;
      description
        "Indicates whether the route got installed in the FIB.";
    }
    leaf route-state {
      type route-state-definition;
      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-definition;
         mandatory true;
         description
           "The reason that causes the route change.";
       }
     }
   }
 }

 <CODE ENDS>
```

4. IANA Considerations

This document registers 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

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a

preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The YANG modules define information that can be configurable in certain instances, for example, a RIB, a route, a nexthop can be created or deleted by client applications, the YANG modules also define RPCs that can be used by client applications to add/delete RIBs, routes and nexthops. In such cases, a malicious client could attempt to remove, add or update a RIB, a route, a nexthop, by creating or deleting corresponding elements in the RIB, route and nexthop lists, respectively. Removing a RIB or a route could lead to disruption or impact in performance of a service, updating a route may lead to suboptimal path and degradation of service levels as well as possibly disruption of service. For those reasons, it is important that the NETCONF access control model is vigorously applied to prevent misconfiguration by unauthorized clients.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability in the ietf-i2rs-rib module:

- RIB: A malicious client could attempt to remove a RIB from a routing instance, for example in order to sabotage the services provided by the RIB, or to add a RIB to a routing instance, hence to inject unauthorized traffic into the nexthop.

- route:A malicious client could attempt to remove or add a route from/to a RIB, for example in order to sabotage the services provided by the RIB.

- nexthop: A malicious client could attempt to remove or add a nexthop from/to RIB, which may lead to suboptimal path and degradation of service levels as well as possibly disruption of service.

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

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-04 (work in progress), December 2017.

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

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

[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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

8.2. Informative References

[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-13 (work in progress), January 2018.

[I-D.ietf-i2rs-usecase-reqs-summary] Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", draft-ietf-i2rs-usecase-reqs-summary-03 (work in progress), November 2016.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <https://www.rfc-editor.org/info/rfc7921>.

Authors' Addresses

Lixing Wang
Individual

Email: wang_little_star@sina.com

Mach(Guoyi) Chen
Huawei

Email: mach.chen@huawei.com

Amit Dass
Ericsson

Email: amit.dass@ericsson.com

Hariharan Ananthakrishnan
Packet Design

Email: hari@packetdesign.com

Sriganesh Kini
Individual

Email: sriganeshkini@gmail.com

Nitin Bahadur
Bracket Computing

Email: nitin_bahadur@yahoo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 17, 2018

N. Bahadur, Ed.
Uber
S. Kini, Ed.

J. Medved
Cisco

February 13, 2018

Routing Information Base Info Model
draft-ietf-i2rs-rib-info-model-14

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 an information model for the RIB to enable defining a standardized data model, and it was used by the IETF's I2RS WG to design the I2RS RIB data model. It is being published to record the higher level informational model decisions for RIBs so that other developers of RIBs may benefit from the design concepts.

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 August 17, 2018.

Copyright Notice

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.

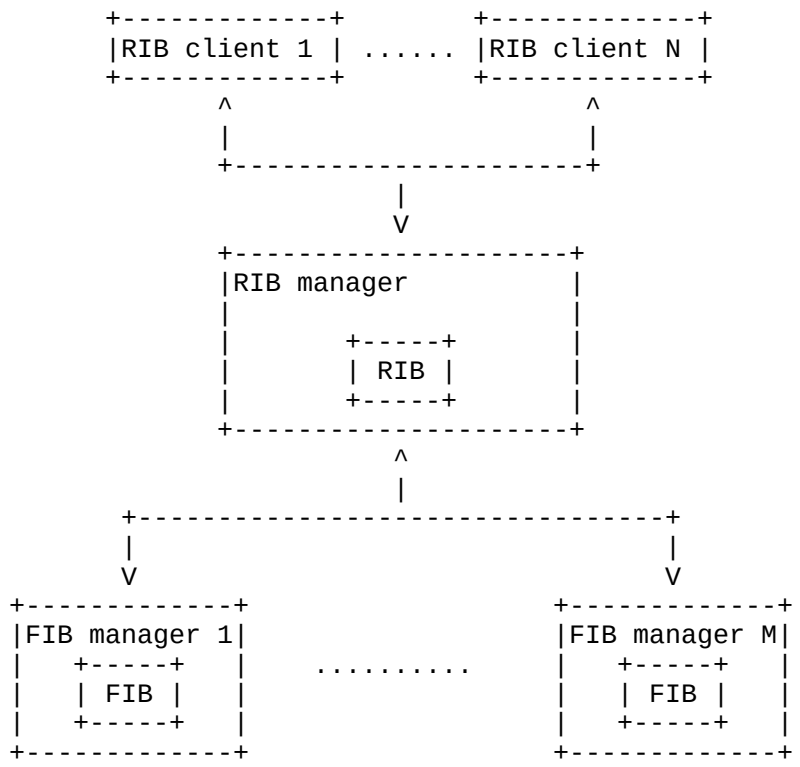


Figure 1: RIB manager, RIB clients and FIB managers

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 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.

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 in Figure 2. Please note that for ease of ASCII art representation this drawing shows a single routing-instance, a single, RIB, and a single route. Sub-sections of this Section describe the logical data nodes that should be contained within a RIB. Section 3 and Section 4 describe the high level read and write operations.

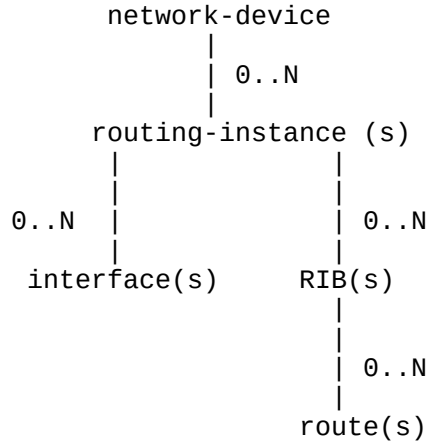


Figure 2: RIB model

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). There MAY be many routing instances and each routing intance MAY contain 1 or more RIBs. 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 MAY have multiple RIBs. A routing instance MAY 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 MAY 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. Please note that for ease of depiction in ASCII art only a single instance of the route attribute, match flags, or nexthop is depicted.

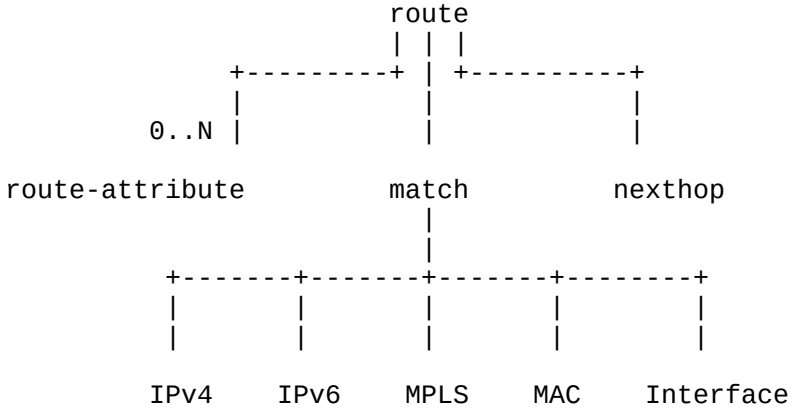


Figure 3: Route model

This document specifies the following match types:

- IPv4: Match on destination and/or source IP address in the IPv4 header
- IPv6: Match on destination and/or source 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

A route MAY be matched on one or more these match types by policy as either an "AND" (to restrict the number of routes) or an "OR" (to combine two filters).

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.

Each route has associated with it a Nexthop. Nexthop is described in Section 2.4.

Additional features to match multicast packets were considered (E.g. TTL of the packet to limit the range of a multicast group), but these were not added to this information model. Future RIB information models should investigate these multicast features.

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. For ease of ASCII art depiction, only a single instance of any component of the nexthop is shown in Figure 4.

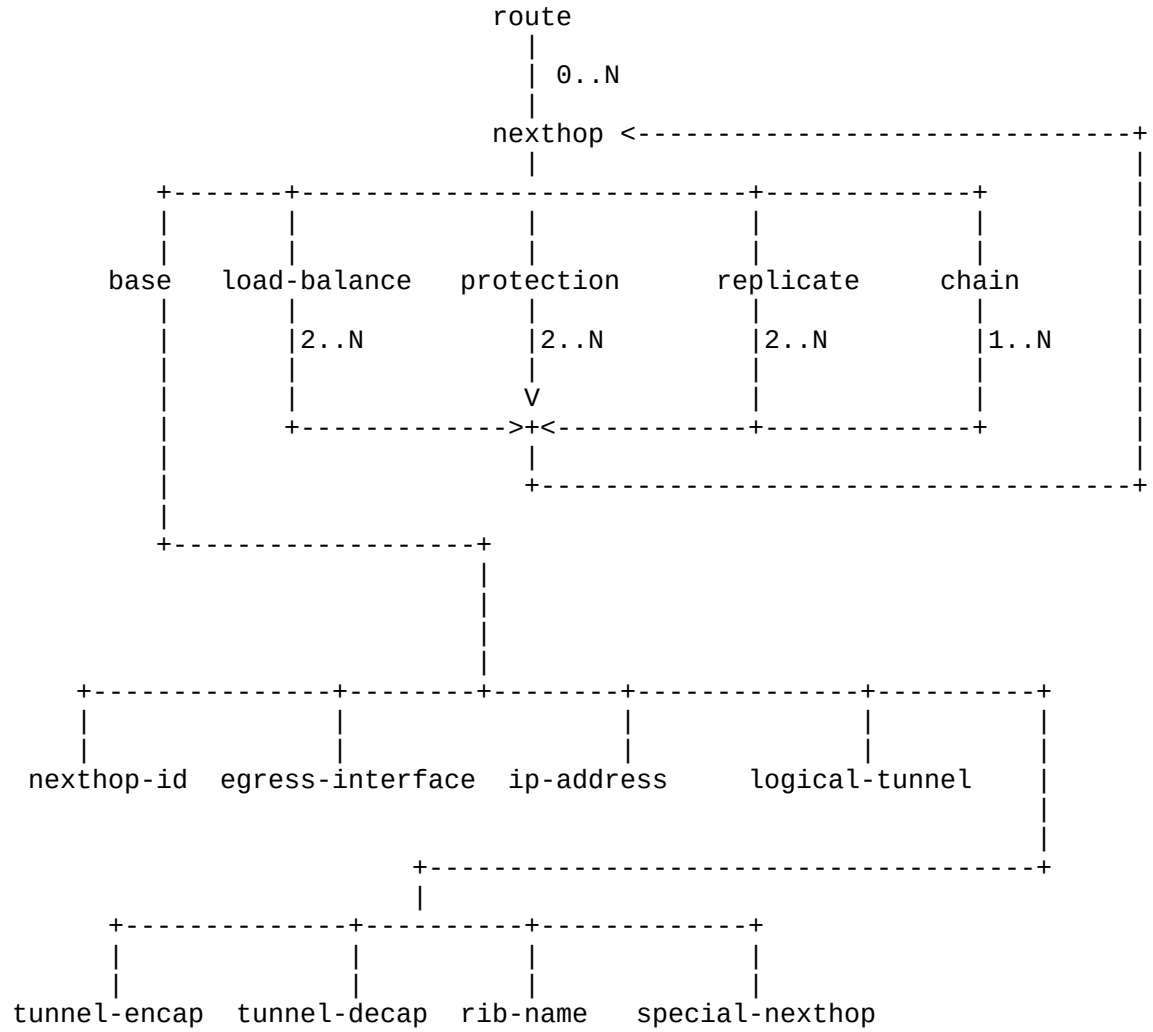


Figure 4: Nexthop model

This document specifies a very generic, extensible and recursive grammar for nexthops. A nexthop can be a base nexthop or a derived nexthop. Section 2.4.1 details base nexthops and Section 2.4.2 explains various kinds of derived nexthops. There are certain special nexthops and those are described in Section 2.4.1.1. Lastly, Section 2.4.3 delves into nexthop indirection and it's use. Examples of when and how to use tunnel nexthops and derived nexthops are shown in Section 7.2.

2.4.1. Base nexthop

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 derived nexthops.
- Interface nexthops - nexthops pointing to an interface. Various attributes associated with these nexthops are:
  - 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.

  These attributes can be used in combination as follows:
  - 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 nexthops - nexthops pointing to a tunnel. Various types of tunnel nexthops are:
  - 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. This indicates that the route lookup needs to continue in the specified RIB. This is a way to perform chained lookups.

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.

2.4.1.1. Special nexthops

Special nexthops are for performing specific well-defined functions (e.g. drop). 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.

2.4.2. Derived nexthops

Derived nexthops can be:

- Weighted lists - for load-balancing
- Preference lists - for protection using primary and backup
- Replication lists - list of nexthops to which to replicate a packet
- Nexthop chains - for chaining multiple operations or attaching multiple headers
- Lists of lists - recursive application of the above

Nexthop chains (See Section 7.2.5 for usage), 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.

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.

2.4.2.1. 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 indirection

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.

One example of usage of indirection is a nexthop that points to another network device (Eg. BGP peer). The returned nexthop identifier can then be used for programming routes to point to the this nexthop. Given that the RIB manager has created an indirection using the nexthop identifier, if the transport path to the network device (BGP peer) changes, that change in path will be seamless to the external entity and all routes that point to that network device 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. See (Section 2.4.2) for examples.

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 can be modified. An object that can be modified 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 Section 2 in order to derive a data model.

```
<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 derived 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 derived 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

The Informational module specified in this document defines a schema

for data models that are designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The RIB info model specifies read and write operations to network devices. These network devices might be considered sensitive or vulnerable in some network environments. Write operations to these network devices without proper protection can have a negative effect on network operations. Due to this factor, it is recommended that data models also consider the following in their design:

- Require utilization of the authentication and authorization features of the NETCONF or RESTCONF suite of protocols.
- Augment the limits on how much data can be written or updated by a remote entity built to include enough protection for a RIB model.
- Expose the specific RIB model implemented via NETCONF/RESTCONF data models.

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,
<https://www.rfc-editor.org/info/rfc2119>.

12.2. Informative References

[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, <https://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, <https://www.rfc-editor.org/info/rfc5120>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[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, <https://www.rfc-editor.org/info/rfc5511>.

[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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

[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, <https://www.rfc-editor.org/info/rfc7920>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Authors' Addresses

Nitin Bahadur (editor)
Uber
900 Arastradero Rd
Palo Alto, CA 94304
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: March 31, 2018

D. Migault
J. Halpern
Ericsson
S. Hares
Huawei

September 27, 2017

I2RS Environment Security Requirements
draft-ietf-i2rs-security-environment-reqs-06

Abstract

This document provides environment security requirements for the I2RS architecture. Environment security requirements are independent of the protocol used for I2RS. The security environment requirements are the good security practices to be used during implementation and deployment of the code related to the new interface to routing system (I2RS) so that I2RS implementations can be securely deployed and operated.

Environmental security requirements do not specify the I2RS protocol seecurity requirements. This is done in another document (draft-ietf-i2rs-protocol-security-requirements).

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 https://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 March 31, 2018.

Copyright Notice

Table of Contents

1. Introduction

This document provides environment security requirements for the I2RS architecture. Environment security requirements are independent of the protocol used for I2RS. The I2RS protocol security requirements [RFC8241] define the security for the communication between I2RS client and agent. The security environment requirements are good security practices to be used during implementation and deployment of the I2RS protocol so that I2RS protocol implementations can be securely deployed and operated. These environment security requirements address the security considerations described in the I2RS Architecture [RFC7921] required to provide a stable and secure environment in which the dynamic programmatic interface to the routing system (I2RS) should operates.

Even though the I2RS protocol is mostly concerned with the interface between the I2RS client and the I2RS agent, the environmental security requirements must consider the entire I2RS architecture and specify where security functions may be hosted and what criteria should be met in order to address any new attack vectors exposed by deploying this architecture. Environment security for I2RS has to be considered for the complete I2RS architecture and not only on the protocol interface.

This document is structured as follows:

- Section 2 describes the terminology used in this document,
- Section 3 describes how the I2RS plane can be securely isolated from the management plane, control plane, and forwarding plane.

The subsequent sections of the document focus on the security within the I2RS plane.

- Section 4 analyses how the I2RS access control policies can be deployed throughout the I2RS plane in order to limit access to the routing system resources to authorized components with the authorized privileges. This analysis examines how providing a robust communication system between the components aids the access control.
- Section 5 details how I2RS keeps applications isolated from another and without affecting the I2RS components. Applications may be independent, with different scopes, owned by different tenants. In addition, the applications may modify the routing system in an automatic way.

Motivations are described before the requirements are given.

The reader is expected to be familiar with the I2RS problem statement [RFC7920], I2RS architecture, [RFC7921], traceability requirements [RFC7922], I2RS Pub/Sub requirements [RFC7923], I2RS ephemeral state requirements [RFC8242], I2RS protocol security requirements [RFC8241].

2. Terminology and Acronyms

- Environment Security Requirements : Security requirements specifying how the environment a protocol operates in needs to be secured. These requirements do not specify the protocol 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: The 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.

2.1. 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].

3. I2RS Plane Isolation

Isolating the I2RS plane from other network planes (the management, forwarding, and control planes) is fundamental to the security of the I2RS environment. Clearly differentiating the I2RS components from the rest of the network device does the following:

1. protects the I2RS components from vulnerabilities in other parts of the network,

2. protects 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 placing software into software containers within modules with clear interfaces to exterior modules. In a similar way, although the I2RS plane cannot be completely isolated from other planes, it can be carefully designed so the interactions between the I2RS plane and other planes can be identified and controlled. The following is a brief description of how the I2RS plane positions itself in regard to the other planes.

3.1. I2RS Plane and Management plane

The purpose of the I2RS plane is to provide a standard programmatic interface to the routing system resources to network oriented applications. Routing protocols often run in a control plane and provide entries for the forwarding plane as shown in figure 1. The I2RS plane contains the I2RS applications, the I2RS client, the north bound interface between the I2RS client and I2RS applications, the I2RS protocol, the I2RS agent, and the south bound API (SB API) to the routing system. The communication interfaces in the I2RS plane are shown on the the left hand side of figure 1.

The management plane contains the mangement application, the management client, the north bound API between the management client and management application, the mangement server, the management protocol (E.g. RESTCONF) between mangement client and management server, and the south bound API between the management server and the control plane. The communication interfaces associated with the management plane are shown on the right hand side of figure 2.

The I2RS plane and the management plane both interact with the control plane on which the routing systems operate. [RFC7921] describes several of these interaction points such as the local configuration, the static system state, routing, and signaling. A routing resource may be accessed by I2RS plane, the mangement plane, or routing protocol(s) which creates the potential for overlapping access. The southbound APIs can limit the scope of the management plane’s and the I2RS plane’s interaction with the routing resources.

Security focus:

Data can be read by I2RS plane from configuration as copy of configuration data, or by management plane as copies of the I2RS plane. The problem is when the I2RS plane installs the routing plane as its new configuration or the management plane installs the I2RS plane information as management plane configuration. In this circumstance, we define "infecting" as interfering with and leading into a incoherent state. Planned interactions such as interactions

denoted in in two cooperating Yang data modules is not an incoherent state.

The primary protection in this space is going to need to be validation rules on:

- the data being sent/received by the I2RS agent (including notification of changes that the I2RS agent sends the I2RS client),
- any data transferred between management datastores (configuration or operational state) and I2RS ephemeral control plane data stores;
- data transferred between I2RS Agent and Routing system,
- data transferred between a management server and the I2RS routing system,
- data transferred between I2RS agent and system (e.g. interfaces ephemeral configuration),
- data transferred between management server and the system (e.g. interface configuration).

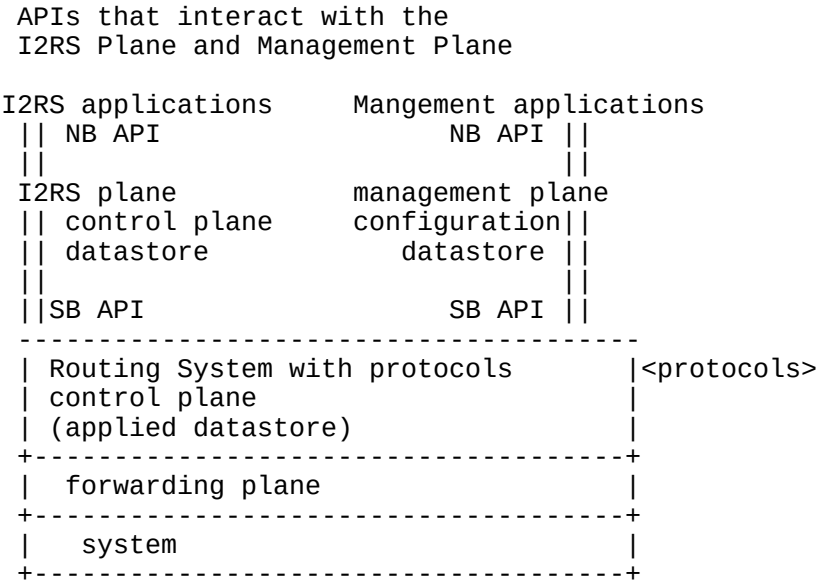


Figure 1 - North Bound (NB) APIs and South Bound (SB) APIs

3.2. I2RS Plane and Forwarding Plane

Applications hosted by the I2RS client belong to the I2RS plane. It is difficult to constrain these applications to the I2RS plane, or even to limit their scope within the I2RS plane. Applications using I2RS may also interact with components outside the I2RS plane. For example an application may use a management client to configure the network and monitored events via an I2RS agent as figure 4 shows.

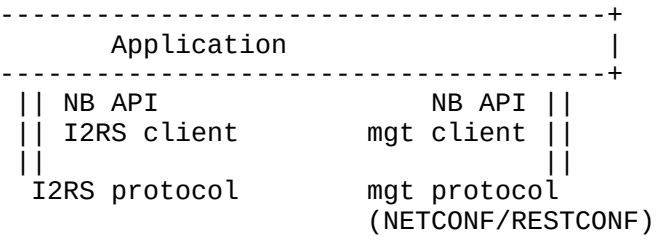


figure 2

Applications may also communicate with multiple I2RS clients in order to have a broader view of the current and potential states of the network and the I2RS plane itself. These varied remote communication

relationships between applications using the I2RS protocol to change the forwarding plane make it possible for an individual application to be an effective attack vector against the operation of the network, a router's I2RS plane, the forwarding plane of the routing system, and other planes (management and control planes).

Prevention measures:

Systems should consider the following prevention errors:

application validation - There is little the I2RS plane can do to validate applications with which it interacts. The I2RS client passes the I2RS agent an unique identifier for the application so that an application's actions can be traced back to the application.

Validation against common misconfigurations or errors - One way of securing the interfaces between application, the I2RS plane, and the forwarding plane is to limit the information accepted and to limit the rate information is accepted between these three software planes. Another method is to perform rudimentary checks on the results of any updates to the forwarding plane.

3.3. 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. I2RS client configures, monitors or receives events via the I2RS agent's interaction with the routing system including the process that handles the control plane signalling protocols (BGP, ISIS, OSPF, etc.), route information databases (RIBs), and interface databases. In some situations, to manage an network outage or to control traffic, the I2RS protocol may modify information in the route database or the configuration of routing process. While this is not a part of normal processing, such action allows the network operator to bypass temporary outages or DoS attacks.

This capability to modify the routing process information carries with it the risk that the I2RS agent may alter the normal properties of the routing protocols which provide normal loop free routing in the control plane. For example, information configured by the I2RS agent into routing process or RIBs could cause forwarding problems or routing loops. As a second example, state which is inserted or deleted from routing processes (control traffic, counters, etc.) could cause the routing protocols to fail to converge or loop).

Prevention measures:

The I2RS implementation can provide internal checks after a routing system protocol change that it is still operating correctly. These checks would be specific to the routing protocol the I2RS Agent would change. For example, if a BGP maximum prefix limit for a BGP peer is lowered then the BGP peer should not allow the number prefixes received from that peer to exceed this number.

3.4. Requirements

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

SEC-ENV-REQ 1: Application-to-routing system resources communications should use an isolated communication channel. Various levels 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 (e.g. using vLAN). In a virtual environment that shares a common infrastructure, encryption may also be used as a way to enforce isolation. Encryption can be added by using a secure transport required by the I2RS protocol security [RFC8241], and sending the non-confidential I2RS data (designed for a non-secure transport) over a secure transport.

SEC-ENV-REQ 2: The interface used by the routing element to receive I2RS transactions via the I2RS protocol (e.g. the IP address) SHOULD be a dedicated physical or logical interface. As previously mentioned, a dedicated physical interface may contribute to a higher isolation. Isolation may also be achieved by using a dedicated IP address or a dedicated port.

SEC-ENV-REQ 3: An I2RS agent SHOULD have specific permissions for interaction with each routing element and access to the routing element should governed by policy specific to the I2RS agent’s interfaces (network, routing system, system, or cross-datastore).

Explanation:

When the I2RS agent performs an action on a routing element, the action is performed in a process (or processes) associated with a routing process. For example, in a typical UNIX system, the user is designated with a user id (uid) and belongs to groups designated by group ids (gid). If such a user id (uid) and group id (gid) is the identifier for the routing processes peforming routing tasks in the control plane, then the I2RS Agent process would communicate with

these routing processes. It is important that the I2RS agent has its own unique identifier and the routing processes have their own identifier so that access control can uniquely filter data between I2RS Agent and routing processes.

The specific policy that the I2RS agent uses to filter data from the network or from different processes on a system (routing, system or cross-datastore) should be specific to the I2RS agent. For example, the network access filter policy that the I2RS agent uses should be uniquely identifiable from the configuration datastore updated by a management protocol.

SEC-ENV-REQ 4: The I2RS plane should be informed when a routing system resource is modified by a user outside the I2RS plane access. Notifications from the control plane SHOULD not be allowed to flood the I2RS plane, and rate limiting (or summarization) is expected to be applied. These routing system notifications MAY translated to the appropriate I2RS agent notifications, and passed to various I2RS clients via notification relays.

Explanation:

This requirements is also described in section 7.6 of [RFC7921] for the I2RS client, and this section extends it to the entire I2RS plane (I2RS agent, client, and application).

A routing system resource may be accessed by management plane or control plane protocols so a change to a routing system resource may remain unnoticed unless and until the routing system resource notifies the I2RS plane by notifying the I2RS agent. Such notification is expected to trigger synchronization of the I2RS resource state between the I2RS agent and I2RS client - signalled by the I2RS agent sending a notification to an I2RS client.

The updated resource should be available in the operational state if there is a yang module referencing that operational state, but this is not always the case. In the cases where operational state is not updated, the I2RS SB (southbound) API should include the ability to send a notification.

SEC-ENV-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 [RFC7921]

Explanation:

A key part of the I2RS architecture is notification regarding routing system changes across the I2RS plane (I2RS client to/from I2RS agent). The security environment requirements above (SEC-ENV-REQ-03 to SEC-ENV-REQ-05) provide the assurance that the I2RS plane and the routing systems the I2RS plane attaches to remains untouched by the other planes or the I2RS plane is notificed of such changes. Section 6.3 of [RFC7921] describes how the I2RS agent within the I2RS plane interacts with forwarding plane's local configuration, and provides the example of an overwrite policy between the I2RS plane and local configuration (instantiated in 2 Policy Knobs) that operators may wish to set. The prompt notification of any outside overwrite is key to the architecture and proper interworking of the I2RS Plane.

4. I2RS Access Control for Routing System Resources

This section provides recommendations on how the I2RS plane's access control should be designed to protect the routing system resources. These security policies for access control only apply within the I2RS plane. More especially, the policies are associated to the applications, I2RS clients and I2RS agents, with their associated identity and roles.

The I2RS deployment of I2RS applications, I2RS clients, and I2RS agents can be located locally in a closed environment or distributed over open networks. The normal case for routing system management is over an open environment. Even in a closed environment, access control policies should be carefully defined to be able to, in the future, carefully extend the I2RS plane to remote applications or remote I2RS clients.

[RFC8241] defines the security requirements of the I2RS protocol between the I2RS client and the I2RS agent over a secure transport. This section focuses on I2RS access control architecture (section 4.1), access control policies of the I2RS agent (section 4.2), the I2RS client (section 4.3), and the application (section 4.4).

4.1. I2RS Access Control Architecture

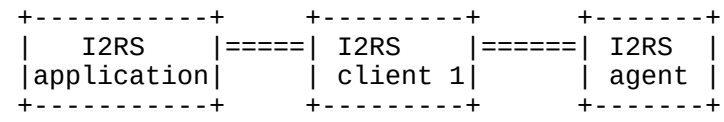
Overview:

Applications access the routing system resource via numerous intermediate nodes. The application communicates with an I2RS client. In some cases, the I2RS client is only associated with a single application attached to one or more agents (case a and case b in figure 4 below). In other cases, the I2RS client may be connected

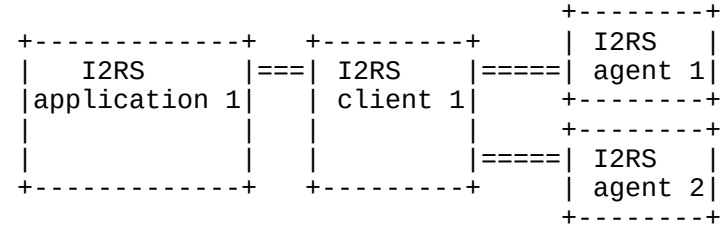
to two applications (case c in figure 4 below), or the I2RS may act as a broker (agent/client device shown in case d in figure 4 below). The I2RS client broker approach provides scalability to the I2RS architecture as it avoids each application being registered to the I2RS agent. Similarly, the I2RS access control should be able to scale to numerous applications.

The goal of the security environment requirements in this section are to control the interactions between the applications and the I2RS client, and the interactions between the I2RS client and the I2RS agent. The key challenge is that the I2RS architecture puts the I2RS Client in control of the stream of communication (application to I2RS client and I2RS client to the I2RS agent). The I2RS agent must trust the I2RS client's actions without having an ability to verify the I2RS client's actions.

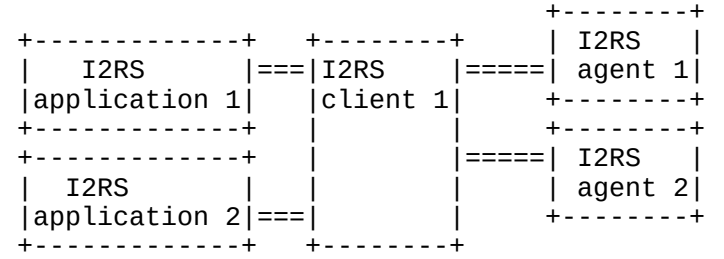
a) I2RS application-client pair talking
to one I2RS agent



b) I2RS application client pair talking to
two i2RS agents



c) two applications talk to 1 client



d) I2RS Broker (agent/client

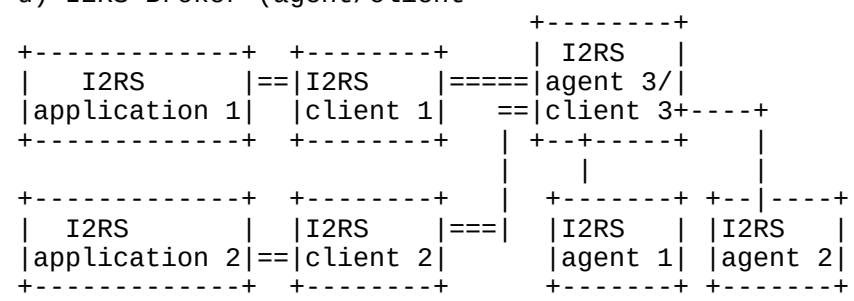


figure 3

4.1.1. Access control Enforcement Scope

SEC-ENV-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. The mechanisms for the I2RS client access control are not in scope of the I2RS architecture [RFC7921], which exclusively focuses on the I2RS agent access control provided by the I2RS protocol.

Explanation:

This architecture 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.

4.1.2. Notification Requirements

SEC-ENV-REQ 7: When an access request to a routing resource is refused by one party (the I2RS client or the I2RS agent), the requester (e.g the application) as well as all intermediaries should indicate the reason the access has not been granted, and which entity rejected the request.

Explanation:

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.

SEC-REQ-07 indicates the I2RS agent may reject the request because the I2RS client is not an authorized I2RS client or lacks the privileges to perform the requested transaction (read, write, start notifications or logging). The I2RS client should be notified of the reason the I2RS agent rejected the transaction due to a lack of authorization or privileges, and the I2RS client should return a message to the application indicating the I2RS agent reject the transaction with an indication of this reason. Similarly, if the I2RS client does not grant the access to the application, the I2RS

client should also inform the application. An example of an error message could be, "Read failure: you do not have 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 relates to the following interactions:

- interactions between the application and the I2RS client,
- interactions between the I2RS client and the I2RS agent at a content level (Protocol security requirements are described by [RFC8241]), and
- interactions between the I2RS agent and the I2RS routing system (forwarding plane, control plane, and routing plane).

4.1.3. Trust

SEC-ENV-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 other, and communication between them should also be trusted (e.g. TLS) in order to reduce additional vectors of attacks.

SEC-ENV-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.

Explanation:

The participants in the I2RS Plane (I2RS client, I2RS agent, and I2RS application) exchange critical information, and to be effective the communication should be trusted and free from security attacks.

The I2RS client or the I2RS agent should be able to reject messages over a communication channel that can be easily hijacked, like a clear text UDP channel.

4.1.4. Sharing access control Information

For the I2RS client:

SEC-ENV-REQ 10: The I2RS client MAY request information on its I2RS access control subset policies from the I2RS agent or cache requests that have been rejected by the I2RS

agent to limit forwarding unnecessary queries to the I2RS agent.

SEC-ENV-REQ 11: The I2RS client MAY support receiving notifications when its I2RS access control subset policies have been updated by the I2RS agent.

Similarly, for the applications:

SEC-ENV-REQ 12: The applications MAY request information on its I2RS access control subset policies in order to limit forwarding unnecessary queries to the I2RS client.

SEC-ENV-REQ 13: The applications MAY subscribe to a service that provides notification when its I2RS access control subset policies have been updated.

For both the application and the client:

SEC-ENV-REQ 14: The I2RS access control should explicitly specify accesses that are granted. More specifically, anything not explicitly granted should be denied (default rule).

Explanation:

In order to limit the number of access requests that result in an error, each application or I2RS client can retrieve the I2RS access control policies that apply 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 mechanisms may be implemented with different levels of completeness and dynamicity of the individual I2RS access control policies. One example may be caching transaction requests that have been rejected.

I2RS access control should be appropriately balanced between the I2RS client and the I2RS agent. It remains relatively easy to avoid the complete disclosure of the access control policies of the I2RS agent. Relative disclosure of access control policies may allow leaking confidential information in case of misconfiguration. It is important to balance the level of trust of the I2RS client and the necessity of distributing the enforcement of the access control policies.

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 requests of the application. 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 benefits and problems with synchronizing its access control policies with the I2RS agent to proxy request validation versus simply passing the access request to the I2RS agent.

- 2) A single I2RS client connects to multiple applications or acts as a broker for many applications:
In this case the I2RS agent has a single I2RS client attached, so the I2RS client could be configured to enforce access control policies instead of the I2RS Agent. In this circumstance, it is possible that the I2RS agent may grant an I2RS client high priviledges and blindly trust the I2RS client without enforcing access control policies on what the I2RS client can do. Such a situation must be avoided as it could be used by malicious applications for a privilege escalation by compromising the I2RS client, causing the I2RS client to perform some action on behalf of the application that it normally does not have the privileges to perform. In order to mitigate such attacks, the I2RS client that connects to multiple applications or operates as a broker is expected to host application with an equivalent level of privileges.

4.1.5. Sharing Access Control in Groups of I2RS Clients and Agents

Overview:

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.

Requirements:

SEC-ENV-REQ 15: I2RS clients should be distributed and act as brokers for applications that share roughly similar permissions.

SEC-ENV-REQ 16: I2RS agents should avoid granting extra privileges to their authorized I2RS client. I2RS agents should be shared by I2RS clients with roughly similar permissions. More explicitly, an I2RS agent shared between I2RS clients that are only provided read access to the routing system resources do not need to perform any write access, so the I2RS client should not be provided these accesses.

SEC-ENV-REQ 17: I2RS clients and I2RS agents should be able to trace [RFC7922] the various transactions they perform as well as suspicious activities. These logs should be collected regularly and analysed by functions that may be out of the I2RS plane.

Explanation:

This restriction for distributed I2RS clients to act as brokers only for applications with roughly the same privileges avoids the I2RS client having extra privileges compared to hosted applications, and discourages applications from performing privilege escalation within an I2RS client. For example, 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 an I2RS agent that is already shared by an I2RS client that requires 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.

The I2RS agent can trace transactions that an I2RS client requests it to perform, and to link this to the application via the secondary opaque identifier to the application. This information is placed in a tracing log which is retrieved by management processes. If a particular application is granted a level of privileges it should not have, then this tracing mechanism may detect this security intrusion after the intrusion has occurred.

4.1.6. Managing Access Control Policy

Access control policies should be implemented so that the policies remain manageable in short and longer term deployments of the I2RS protocol and the I2RS plane.

Requirements:

SEC-ENV-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 (I2RS client, I2RS agent, and application).

Explanation:

Granting or configuring an application with new policy should not require manual configuration of I2RS clients, I2RS agents, or other applications.

SEC-ENV-REQ 19: Access control should be scalable when the number of application grows as well as when the number of I2RS clients increases.

Explanation:

A typical implementation of a local I2RS client access control policies may result in creating manually a system user associated with each application. Such an approach is not likely to scale when the number of applications increases into the hundreds.

SEC-ENV-REQ 20: Access control should be dynamically managed and easily updated.

Explanation:

Although the number of I2RS clients is expected to be lower than the number of applications, as I2RS agents provide access to the routing resource, it is of primary importance that an access can be granted or revoke in an efficient way.

SEC-ENV-REQ 21: I2RS clients and I2RS agents should be uniquely identified in the network to enable centralized management of the I2RS access control policies.

Explanation:

Centralized management of the access control policies of an I2RS plane with network that hosts several I2RS applications, clients and agents requires that each devices can be identified.

4.2. I2RS Agent Access Control Policies

Overview:

The I2RS agent access control restricts 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.

Second, 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.

Third, each I2RS client may contact various I2RS agent with different privileges and access control policies.

4.2.1. I2RS Agent Access Control

This section provides recommendations on the I2RS agent access control policies to keep I2RS access control coherent within the I2RS plane.

Requirements:

SEC-ENV-REQ 22: I2RS agent access control policies should be primarily based on the I2RS clients as described in [RFC7921].

SEC-ENV-REQ 23: I2RS agent access control policies MAY be based on the application if the application identity has been authenticated by the I2RS client and passed via the secondary identity to the I2RS agent.

SEC-ENV-REQ 24: The I2RS agent should know which identity (E.g. 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.

SEC-ENV-REQ 25: the I2RS agent should have an "I2RS agent overwrite Policy" that indicates how identities can be prioritized. This requirement is also described in section 7.6 of [RFC7921]. Similar requirements exist for components within the I2RS plane, but this is within the scope of the I2RS protocol security requirements [RFC8241].

Explanation:

If the I2RS application is authenticated to the I2RS client, and the I2RS client is authenticated to the I2RS agent, and the I2RS client uses the opaque secondary identifier to pass an authenticated identifier to the I2RS agent, then this identifier may be used for access control. However, caution should be taken when using this chain of authentication since the secondary identifier is intended in the I2RS protocol only to aid traceability.

From the environment perspective the I2RS agent MUST be aware when the resource has been modified outside the I2RS plane by another plane (management, control, or forwarding). The prioritization between the different planes should set a deterministic policy that allows the collision of two planes (I2RS plane and another plane) to be resolved via an overwrite policy in the I2RS agent.

Similar requirements exist for knowledge about identities within the I2RS plane which modify things in the routing system, but this is within the scope of the I2RS protocol’s requirements for ephemeral state [RFC8242] and security requirements [RFC8241].

4.2.2. I2RS Client Access Control Policies

Overview:

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.

Requirements:

REQ 26: I2RS client should authenticate its applications. If the I2RS client acts as a broker and supports multiple applications, it should authenticate each application.

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 immediately and transparently 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.

Explanation:

If no authentication mechanisms have being provided between the I2RS client and the application, then the I2RS client must be dedicated to a single application. By doing so, application authentication relies on the I2RS authentication mechanisms between the I2RS client and the I2RS agent.

If an I2RS client has multiple identities that are associated with different privileges for accessing an I2RS agent(s), the I2RS client access control policies should specify the I2RS client identity with the access control policy.

4.2.3. Application and Access Control Policies

Overview

Applications do 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.

Requirements:

SEC-ENV-REQ 28: Applications SHOULD be uniquely identified by their associated I2RS clients

Explanation:

Different application may use different methods (or multiple methods) to communicate with its associated I2RS client, and each application may not use the same form of an application identifier. However, the I2RS client must obtain an identifier for each application. One method for this identification can be a system user id.

SEC-ENV-REQ 29: Each application SHOULD be associated to a restricted number of I2RS clients.

Explanation:

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. This does not prevent an I2RS client to host a large number of applications with the same levels of trust.

SEC-ENV-REQ 30: An application SHOULD be provided means and methods to contact their associated I2RS client.

Explanation:

It is obvious when an I2RS client belongs to the application as part of a module or a library that the application can communicate with a I2RS client. Similarly, if the application runs into a dedicated system with a I2RS client, it is obvious which I2RS client the application should contact. If the application connects to the I2RS client remotely, the application needs some means to retrieve the necessary information to contact its associated I2RS client (e.g. an IP address or a FQDN).

5. I2RS Application Isolation

A key aspect of the I2RS architecture is the network oriented application that uses the I2RS high bandwidth programmatic interface to monitor or change one or more routing systems. I2RS applications could be control by a single entity or serve various tenants of the network. If multiple entities use an I2RS application to monitor or change the network, security policies must preserve the isolation of each entity’s control and not let malicious entities controlling one I2RS application interfere with other I2RS applications.

This section discusses both security aspects related to programmability as well as application isolation in the I2RS architecture.

5.1. Robustness Toward Programmability

Overview

I2RS provides a programmatic interface in and out of the Internet routing system which provides the following advantages for security:

- the use of automation reduces configuration errors;

- the programmatic interface enables fast network reconfiguration and agility in adapting to network attacks; and
- monitoring facilities to detect a network attack, and configuration changes which can help mitigate the network attack.

Programmability allows applications to flexible control which may cause problems due to:

- applications which belong to different tenants with different objectives,
- applications which lack coordination resulting in unstable routing configurations such as oscillations between network configurations, and creation of loops. For example, one application may monitor a state and change to positive, and a second application performs the reverse operation (turns it negative). This fluctuation can cause a routing system to become unstable.

The I2RS plane requires data and application isolation to prevent such situations from happening. However, to guarantee the network stability constant monitoring and error detection are recommended.

Requirement:

SEC-ENV-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 any I2RS clients or applications causing problems that may leave the routing system in an unstable state.

Explanation:

In the least, monitoring consists of logging events and receiving streams of data. I2RS Plane implementations should monitor the I2RS applications and I2RS clients for potential problems. The cause for the I2RS clients or applications providing problematic requests can be failures in the implementation code or malicious intent.]

5.2. Application Isolation

5.2.1. DoS

Overview:

Requirements for robustness to DoS attacks have been addressed in the communication channel section [RFC7921]. This section focuses on requirements for application isolation that help prevent DoS.

Requirements:

SEC-ENV-REQ 32: In order to prevent DoS, it is recommended the I2RS agent control the resources allocated to each I2RS client. I2RS clients that act as broker may not be protected efficiently against these attacks unless the broker performs resource controls for the hosted applications.

SEC-ENV-REQ 33: I2RS agent SHOULD not make a response redirection unless the redirection is previously validated and agreed by the destination.

SEC-ENV-REQ 34: I2RS Applications should avoid the use of underlying protocols that are not robust enough to protect against reflection attacks.

Explanation:

The I2RS interface is used by applications to interact with the routing states. If the I2RS client is shared between multiple applications, one application can use the I2RS client to perform DoS or DDoS attacks on the I2RS agent(s) and through the I2RS agents attack the network. DoS attack targeting the I2RS agent would consist in providing requests that keep the I2RS agent busy for a long time. These attacks on the I2RS agent may involve an application (requesting through an I2RS Client) heavy computation by the I2RS agent in order to block operations like disk access.

Some DoS attacks may attack the I2RS Client’s reception of notification and monitoring data streams over the network. Other DoS attacks may focus on the application directly by performing reflection attacks to reflect traffic. Such an attack could be performed by first detecting an application is related to monitoring the RIB or changing the RIB. Reflection-based DoS may also attack at various levels in the stack utilizing UDP at the service to redirect data to a specific repository

I2RS implementation should consider how to protect I2RS against such attacks.

5.2.2. Application Logic Control

Overview

This section examines how application logic must be designed to ensure application isolation.

Requirements:

SEC-ENV-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.

Explanation:

Applications use the I2RS interface in order to update the routing system. These updates may be driven by behaviour on the forwarding plane or any external behaviours. In this case, correlating observation with the I2RS traffic may enable the derivation 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.

6. Security Considerations

This whole document is about security requirements for the I2RS environment. To protect personal privacy, any identifier (I2RS application identifier, I2RS client identifier, or I2RS agent identifier) should not contain personal identifiable information.

7. IANA Considerations

No IANA considerations for this requirements.

8. Acknowledgments

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

9. References

9.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[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, <https://www.rfc-editor.org/info/rfc7920>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <https://www.rfc-editor.org/info/rfc7921>.

[RFC7922] Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", RFC 7922, DOI 10.17487/RFC7922, June 2016, <https://www.rfc-editor.org/info/rfc7922>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <https://www.rfc-editor.org/info/rfc7923>.

[RFC8241] Hares, S., Migault, D., and J. Halpern, "Interface to the Routing System (I2RS) Security-Related Requirements", RFC 8241, DOI 10.17487/RFC8241, September 2017, <https://www.rfc-editor.org/info/rfc8241>.

9.2. Informative References

[RFC8242] Haas, J. and S. Hares, "Interface to the Routing System (I2RS) Ephemeral State Requirements", RFC 8242, DOI 10.17487/RFC8242, September 2017, <https://www.rfc-editor.org/info/rfc8242>.

[I-D.ietf-netconf-rfc6536bis] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-05 (work in progress), September 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-04 (work in progress), August 2017.

Authors' Addresses

Daniel Migault
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

I2RS Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 15, 2018

Y. Zhuang
D. Shi
Huawei
R. Gu
China Mobile
H. Ananthakrishnan
Packet Design

February 11, 2018

A YANG Data Model for Fabric Topology in Data Center Networks
draft-ietf-i2rs-yang-dc-fabric-network-topology-06

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 https://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 August 15, 2018.

Copyright Notice

Table of Contents

1. Introduction

Normally, a data center (DC) network is composed of single or multiple fabrics which are also known as PODs (Points Of Delivery). These fabrics may be heterogeneous due to implementation of different technologies when a DC network is upgraded or new techniques and features are enrolled. For example, Fabric A may use VXLAN while Fabric B may use VLAN within a DC network. Likewise, an existing fabric may use VXLAN while a new fabric, for example a fabric introduced for DC upgrade and expansion, may implement a technique discussed in NVO3 WG, such as Geneve [I-D. draft-ietf-nvo3-geneve]. The configuration and management of such DC networks with heterogeneous fabrics will result in considerable complexity, requiring a fair amount of sophistication.

Luckily, for a DC network, a fabric can be considered as an atomic structure for management purposes. From this point of view, the management of the DC network can be decomposed into a set of tasks to manage each fabric separately, as well as the fabric interconnections. This way, the overall management task becomes very flexible and makes it easy to expand and adopt to DC networks that evolve over time.

As a basis for DC fabric management, this document defines a YANG data model [6020][7950] for fabric-based data center topology. To do so, it augments the generic network and network topology data models defined in [I-D.ietf-i2rs-yang-network-topo] with information that is specific to Data Center fabric networks.

The model defines the generic configuration and operational state for a fabric-based network topology, which can subsequently be extended by vendors with vendor-specific information as needed. The model can be used by a network controller to represent its view of the fabric topology that it controls and expose this view to network administrators or applications for DC network management.

Within 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 the 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 the fabric layer to elements of other topologies, such as L3 topologies as defined in [I.D. draft-ietf-i2rs-yang-l3-topology].

By using the fabric topology model defined in this document, people can treat a fabric as a holistic entity and focus on characteristics of a fabric (such as encapsulation type, gateway type, etc.) as well as its connections to other fabrics while putting the underlay topology aside. As such, clients can consume the topology information at the fabric level with no need to be aware of the entire set of links and nodes in the corresponding underlay networks. A fabric topology can be configured by a network administrator using the controller by adding physical devices and links into a fabric. Alternatively, fabric topology can be learned 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]. 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.

2.1. Terminology

Fabric: also known as a POD, is a module of network, compute, storage, and application components that work together to deliver networking services. It represents a repeatable design pattern. Its

components maximize the modularity, scalability, and manageability of data centers.

3. Model Overview

This section provides an overview of the data center 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.

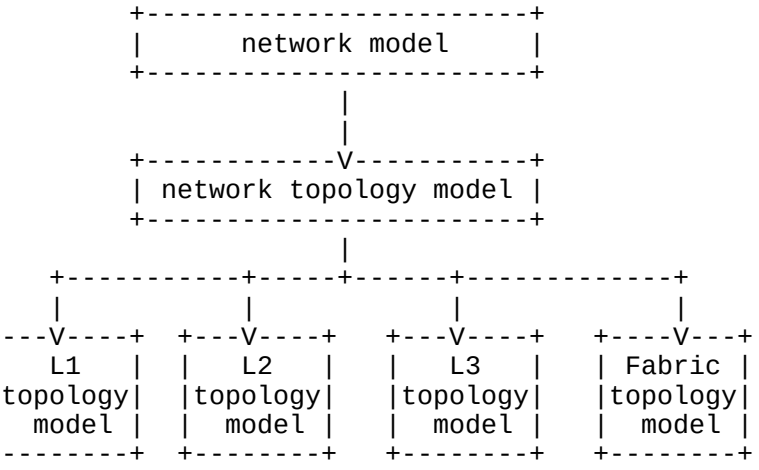


Figure 1: The network data model structure

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. The main purpose of this module is to configure and manage fabrics and their connections. It provides a fabric-based topology view for data center applications.

3.2.1. Fabric Topology

In the fabric topology module, a fabric is modeled as a node of a network, as such the fabric-based data center network consists of a set of fabric nodes and their connections. The following depicts a snippet of the definitions to show the main structure of the model. The notation syntax follows [I-D.draft-ietf-netmod-yang-tree-diagrams].

```
module: ietf-fabric-topology
augment /nw:networks/nw:network/nw:network-types:
   +--rw fabric-network!
augment /nw:networks/nw:network/nw:node:
   +--rw fabric-attributes
          +--rw fabric-id?              fabric-id
          +--rw name?           string
          +--rw type?           fabrictype:underlay-network-type
          +--rw description?    string
          +--rw options
          +--...
augment /nw:networks/nw:network/nw:node/nt:termination-point:
   +--ro fport-attributes
          +--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 "fabric-attributes" is defined to carry attributes for a fabric such as gateway mode, fabric types, involved device nodes, and links.
- Termination points (in network topology module) are augmented with fabric port attributes defined in a container. The "termination-point" here is used to represent a fabric "port" that provides connections to other nodes, such as an internal device, another fabric externally, or end hosts.

Details of the fabric node and the fabric termination point extension will be explained in the following sections.

3.2.2. Fabric node extension

As an atomic network, a fabric itself is composed of a set of network elements i.e. devices, and related links. The configuration of a fabric is contained under the "fabric-attributes" container depicted as follows. The notation syntax follows [I-D.draft-ietf-netmod-yang-tree-diagrams].

```
               +--rw fabric-attributes
                  +--rw fabric-id?      fabric-id
                  +--rw name?           string
                  +--rw type?           fabrictype:underlay-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-capabilit
ies
                  +--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?   fabrictypes:port-type
                         +--rw bandwidth?   fabrictypes:bandwidth
```

In the module, additional data objects for fabric 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 the gateway-mode type (centralized or distributed) and traffic-behavior (whether an Access Control Lists (ACLs) is needed for the traffic). Also, it includes a list of device-nodes and related links as supporting-nodes to form a fabric network. These device nodes and links are represented as leaf-refs of existing nodes and links in the underlay topology. For the device-node, the "role" object is defined to represent the role of a device within the fabric, such as "SPINE" or "LEAF", which should work together with the gateway-mode.

3.2.3. Fabric termination-point extension

Since a fabric can be considered as a node, "termination-points" can represent fabric "ports" that connect to other fabrics, end hosts, as well as devices inside the fabric.

As such, the set of "termination-points" of a fabric indicate all connections of the fabric, including its internal connections, interconnections with other fabrics, and connections to end hosts.

The structure of fabric ports is as follows. The notation syntax follows [I-D.draft-ietf-netmod-yang-tree-diagrams].

The structure of fabric ports is as follows:

```
        augment /nw:networks/nw:network/nw:node/nt:termination-point:
              +--ro fport-attributes
                 +--ro name?          string
                 +--ro role?          fabric-port-role
                 +--ro type?          fabric-port-type
                 +--ro device-port?   tp-ref
                 +--ro (tunnel-option)?
```

It augments the termination points (in network topology module) with fabric port attributes defined in a container.

New nodes are defined for fabric ports including fabric 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). By defining the device-port as a tp-ref, a fabric port can be mapped to a device node in the underlay network.

Also, a new container for tunnel-options is introduced to present the tunnel configuration on a port.

The termination point information is learned from the underlay networks, not configured by the fabric topology layer.

4. Fabric YANG Module

```
<CODE BEGINS> file "ietf-dc-fabric-types@2017-12-21.yang"
    module ietf-dc-fabric-types {

    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-dc-fabric-types";
    prefix fabrictypes;
```

```
       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>

     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-ietf-i2rs-yang-dc-fabric-network-topology;
        see the RFC itself for full legal notices.

        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-dc-fabric-network-topology-03 with RFC
        number when published (i.e. RFC xxxx).";

    revision "2017-12-21"{
        description
            "Initial revision.
             NOTE TO RFC EDITOR: Please replace the following reference to
             draft-ietf-i2rs-yang-dc-fabric-network-topology-03 with RFC
             number when published (i.e. RFC xxxx).";
        reference
            "draft-ietf-i2rs-yang-dc-fabric-network-topology-03";
        }

    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";
    }

    identity trill-fabric {
        base fabric-type;
        description "Trill fabric";
    }
    identity port-type {
        description
           "Base type for fabric port";
        }
    identity eth {
        base port-type;
        description "ETH";
    }
    identity serial {
        base port-type;
        description "Serial";
    }

    identity bandwidth {
        description "Base for bandwidth";
    }
    identity bw-1M {
        base bandwidth;
        description "1M";
    }
    identity bw-10M {
        base bandwidth;
        description "10M";
    }
    identity bw-100M {
        base bandwidth;
        description "100M";
    }
    identity bw-1G {
        base bandwidth;
        description "1G";
    }
    identity bw-10G {
```

```
        base bandwidth;
        description "10G";
    }
    identity bw-40G {
        base bandwidth;
        description "40G";
    }
    identity bw-100G{
        base bandwidth;
        description "100G";
    }

    identity device-role {
        description "Base for the device role in a fabric.";
    }
    identity spine {
        base device-role;
        description "This is a spine node in a fabric.";
    }
    identity leaf {
        base device-role;
        description "This is a leaf node in a fabric. ";
    }
    identity border {
        base device-role;
        description "This is a border node to connect to
        other fabric/network.";
    }
    identity fabric-port-role {
        description "Base for the port's role in a fabric.";
    }
    identity internal {
        base fabric-port-role;
        description "The port is used for devices to access
        each other within a fabric.";
    }
    identity external {
        base fabric-port-role;
        description "The port is used for a fabric to connect
        to outside network.";
    }
    identity access {
        base fabric-port-role;
        description "The port is used for an endpoint to
        connect to a fabric.";
    }

    /*
```

```
     * Typedefs
     */
    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";
    }

    typedef port-type {
        type identityref {
             base port-type;
        }
        description "Port type: ethernet or serial or others.";
    }
    typedef bandwidth {
        type identityref {
             base bandwidth;
        }
        description "Bandwidth of the port.";
    }
    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 underlay-network-type {
        type identityref {
```

```
            base fabric-type;
        }
        description "The type of physical network that implements this
        fabric.Examples are vlan, and trill.";
    }
    typedef device-role {
         type identityref {
            base device-role;
        }
        description "Role of the device node.";
    }
    typedef fabric-port-role {
        type identityref {
            base fabric-port-role;
        }
        description "Role of the port in a fabric.";
    }

    typedef fabric-port-type {
        type enumeration {
            enum layer2interface {
                description "L2 interface";
            }
            enum layer3interface {
                description "L3 interface";
            }
            enum layer2Tunnel {
                description "L2 tunnel";
            }
            enum layer3Tunnel {
                description "L3 tunnel";
            }
        }
                description
            "Fabric port type";
    }

    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 to connect two fabrics.
            It could be L2 Tunnel or L3 Tunnel.";
        }
    }
}
<CODE ENDS>

<CODE BEGINS> file "ietf-dc-fabric-topology@2018-02-11.yang"
    module ietf-dc-fabric-topology {

    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-dc-fabric-topology";
    prefix fabric;

    import ietf-network {
        prefix nw;

        reference
        "draft-ietf-i2rs-yang-network-topo-20
        NOTE TO RFC EDITOR:
        (1) Please replace above reference to
        draft-ietf-i2rs-yang-network-topo-20 with RFC
        number when published (i.e. RFC xxxx).
        (2) Please replace the date in the revision statement with the
         date of publication when published.";
    }

    import ietf-network-topology {
        prefix nt;

        reference
        "draft-ietf-i2rs-yang-network-topo-20
        NOTE TO RFC EDITOR:
        (1) Please replace above reference to
        draft-ietf-i2rs-yang-network-topo-20 with RFC
        number when published (i.e. RFC xxxx).
        (2) Please replace the date in the revision statement with the
         date of publication when published.";
    }
```

```
    import ietf-dc-fabric-types {
        prefix fabrictypes;

        reference
        "draft-ietf-i2rs-yang-dc-fabric-network-topology-03
        NOTE TO RFC EDITOR:
         (1) Please replace above reference to draft-ietf-i2rs-yang-dc
         -fabric-network-topology-03 with RFC number when publised
         (i.e. RFC xxxx).
         (2) Please replace the data in the revision statement with the
         data of publication when published.";
    }

    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>

     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-ietf-i2rs-yang-dc-fabric-network-topology;see the RFC
        itself for full legal notices.

        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-dc-fabric-network-topology-03 with RFC
        number when published (i.e. RFC xxxx).";
```

```
        revision "2018-02-11"{
        description
             "Initial revision.
              NOTE TO RFC EDITOR: Please replace the following reference
              to draft-ietf-i2rs-yang-dc-fabric-network-topology-03 with
              RFC number when published (i.e. RFC xxxx).";
        reference
            "draft-ietf-i2rs-yang-dc-fabric-network-topology-05";
    }

    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 "The Fabric uses centerilized gateway, in
                    which gateway is deployed on SPINE node.";
                }
                enum distributed {
                    description "The Fabric uses distributed gateway, in
                    which gateway is deployed on LEAF node.";
                }
            }
            default "distributed";
            description "Gateway mode of the fabric";
```

```
        }

        leaf traffic-behavior {
            type enumeration {
                enum normal {
                    description "Normal, no policy is enforced.";
                }
                enum policy-driven {
                    description "Policy driven";
                }
            }
            default "normal";
            description "Traffic behavior of the fabric";
        }

        leaf-list capability-supported {
            type fabrictypes:service-capabilities;
            description
               "Supported services of the fabric";
        }
    }

    grouping device-attributes {
        description "device attributes";
        leaf device-ref {
            type fabrictypes:node-ref;
            description
                "The device the fabric includes.";
        }
        leaf-list role {
            type fabrictypes:device-role;
            default fabrictypes:leaf;
            description
                "Role of the device node";
        }
    }

    grouping link-attributes {
                description "Link attributes";
        leaf link-ref {
            type fabrictypes:link-ref;
            description
                "The link it includes";
        }
    }

    grouping port-attributes {
        description "Port attributes";
```

```
        leaf port-ref {
            type fabrictypes:tp-ref;
            description
                "The port it refers to.";
        }
        leaf port-type {
            type fabrictypes:port-type;
            description
                "Port type: ethernet or serial or others.";
        }
        leaf bandwidth {
            type fabrictypes:bandwidth;
            description
                "Bandwidth of 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 fabrictypes:underlay-network-type;
            description
                "The type of physical network that implements this
                 fabric.Examples are vlan, and trill.";
        }

        container vni-capacity {
            description "Number of vni(VXLAN Network Identifier, see [RFC7348])s
 that 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 "Device nodes that include in a fabric.";
            uses device-attributes;
        }

        list device-links {
            key link-ref;
            description "Links that include within a fabric.";
            uses link-attributes;
        }

        list device-ports {
            key port-ref;
            description "Ports that include in 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:fabric-network" {
```

```
        description
          "Augmentation parameters apply only for networks
           with fabric topology";
      }
        description "Augmentation for fabric nodes created by fabric topology.";

        container fabric-attributes {
                        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:fabric-network" {
        description
          "Augmentation parameters apply only for networks
           with fabric topology";
      }
        description "Augmentation for port on fabric.";

        container fport-attributes {
            config false;
            description
               "Attributes for fabric ports";
            uses fabrictypes:fabric-port;
        }
    }
}
<CODE ENDS>
```

5. IANA Considerations

This document registers the following namespace URIs in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-dc-fabric-types Registrant Contact: The IESG. XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-dc-fabric-topology Registrant Contact: The IESG. XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-dc-fabric-topology-state Registrant Contact: The IESG. XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG modules in the "YANG Module Names" registry [RFC6020]:

NOTE TO THE RFC EDITOR: In the list below, please replace references to "draft-ietf-i2rs-yang-dc-fabric-network-topology-03 (RFC form)" with RFC number when published (i.e. RFC xxxx).

Name: ietf-dc-fabric-types Namespace: urn:ietf:params:xml:ns:yang:ietf-dc-fabric-types Prefix: fabrictypes Reference: draft-ietf-i2rs-yang-dc-fabric-network-topology-03.txt (RFC form)

Name: ietf-dc-fabric-topology Namespace: urn:ietf:params:xml:ns:yang:ietf-dc-fabric-topology Prefix: fabric Reference: draft-ietf-i2rs-yang-dc-fabric-network-topology-03.txt (RFC form)

Name: ietf-dc-fabric-topology-state Namespace: urn:ietf:params:xml:ns:yang:ietf-dc-fabric-topology-state Prefix: sfabric Reference: draft-ietf-i2rs-yang-dc-fabric-network-topology-03.txt (RFC form)

6. Security Considerations

The YANG module defined in this document is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content. The subtrees and data nodes and their sensitivity/vulnerability in the ietf-dc-fabric-topology module are as follows:

fabric-attributes: A malicious client could attempt to sabotage the configuration of important fabric attributes, such as device-nodes or type.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. The subtrees and data nodes and their sensitivity/vulnerability in the ietf-dc-fabric-topology module are as follows:

fport-attributes: A malicious client could attempt to read the connections of fabrics without permission, such as device-port, name.

7. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Alexander Clemm, Donald E. Eastlake, Xufeng Liu, Susan Hares, Wei Song, Luis M. Contreras and Benoit Claise.

8. References

8.1. 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-netmod-revised-datastores-06]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", I-D draft-ietf-netmod-revised-datastores-06, October 2017.

[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>.

[RFC5246] Dierks, T. and E. Rescorla, "Transport Layer Security (TLS) Protocol Version 1.2", August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", June 2011, <http://www.rfc-editor.org/info/rfc6242>.

[RFC6536] Bierman, A. and B. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", March 2012, <http://www.rfc-editor.org/info/rfc6536>.

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

[RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", August 2014, <http://www.rfc-editor.org/info/rfc7348>.

[RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, Auguest 2016.

[RFC8040] Bierman, A., Bjorklund, B., and K. Watsen, "RESTCONF Protocol", Jan 2017, <http://www.rfc-editor.org/info/rfc8040>.

8.2. 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-netmod-yang-tree-diagrams, May 2015.

[I-D.draft-ietf-netmod-yang-tree-diagrams] Bjorklund, M. and L. Berger, "YANG Tree Diagrams", I-D draft-ietf-netmod-yang-tree-diagrams, October 2017.

[I-D.draft-ietf-nvo3-geneve] Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", I-D draft-ietf-nvo3-geneve-05, September 2017.

Appendix A. Non NMDA -state modules

The YANG module ietf-fabric-toplogy defined in this document augments two modules, ietf-network and ietf-network-topology, that are designed to be used in conjunction with implementations that support the Network Management Datastore Architecture (NMDA) defined in [I-D.draft-ietf-netmod-revised-datastores]. In order to allow

implementations to use the model even in case when NMDA is not supported, a set of companion modules have been defined that represent a state model of networks and network topologies, ietf-network-state and ietf-network-topology-state, respectively.

In order to be able to use the model for fabric topologies defined in this in this document in conjunction with non-NMDA compliant implementations, a corresponding companion module needs to be introduced as well. This companion module, ietf-fabric-topology-state, mirrors ietf-fabric-topology. However, the module augments ietf-network-state (instead of ietf-network and ietf-network-topology) and all of its data nodes are non-configurable.

Like ietf-network-state and ietf-network-topology-state, ietf-fabric-topology-state SHOULD NOT be supported by implementations that support NMDA. It is for this reason that the module is defined in the Appendix.

The definition of the module follows below. As the structure of the module mirrors that of its underlying module, the YANG tree is not depicted separately.

```
 <CODE BEGINS> file "ietf-dc-fabric-topology-state@2018-02-11.yang"
 module ietf-dc-fabric-topology-state {

 yang-version 1.1;
 namespace "urn:ietf:params:xml:ns:yang:ietf-dc-fabric-topology-state";
 prefix sfabric;

 import ietf-network-state {
     prefix nws;
     reference
     "draft-ietf-i2rs-yang-network-topo-20
     NOTE TO RFC EDITOR:
     (1) Please replace above reference to
     draft-ietf-i2rs-yang-network-topo-20 with RFC
     number when published (i.e. RFC xxxx).
     (2) Please replace the date in the revision statement with the
      date of publication when published.";
 }
 import ietf-dc-fabric-types {
     prefix fabrictypes;

     reference
     "draft-ietf-i2rs-yang-dc-fabric-network-topology-03
     NOTE TO RFC EDITOR:
     (1) Please replace above reference to draft-ietf-i2rs-yang-dc
     -fabric-network-topology-03 with RFC number when published
```

```
        (i.e. RFC xxxx).
        (2) Please replace the data in the revision statement with the
        data of publication when published.";
    }
    import ietf-dc-fabric-topology {
        prefix fabric;

        reference
        "draft-ietf-i2rs-yang-dc-fabric-network-topology-03
         NOTE TO RFC EDITOR:
         (1) Please replace above reference to draft-ietf-i2rs-yang-dc
         -fabric-network-topology-03 with RFC number when published
         (i.e. RFC xxxx).
         (2) Please replace the data in the revision statement with the
         data of publication when published.";
    }

    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>

     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 state, representing topology that is either
        learned, or topology that results from applying toplogy
        that has been configured per the ietf-dc-fabric-topology
        model, mirroring the corresponding data nodes in this model.

        This model mirrors the configuration tree of
        ietf-dc-fabric-topology, but contains only read-only state
        data. The model is not needed when the implemtation
        infrastructure supports the Network Management Datastore
        Architecture(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-ietf-i2rs-yang-dc-fabric-network-topology;see the RFC
        itself for full legal notices.

        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-dc-fabric-network-topology-03 with RFC number
        when published (i.e. RFC xxxx).";

    revision "2018-02-11" {
        description
            "Initial revision.
             NOTE TO RFC EDITOR: Please replace the following reference to
             draft-ietf-i2rs-yang-dc-fabric-network-topology-03 with RFC
             number when published (i.e. RFC xxxx).";
        reference
            "draft-ietf-i2rs-yang-dc-fabric-network-topology-05";
    }

    //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 "The Fabric uses centerilized gateway, in which
            gateway is deployed on SPINE node.";
          }
          enum distributed {
            description "The Fabric uses distributed gateway, in which
            gateway is deployed on LEAF node.";
          }
        }
        default "distributed";
        description "Gateway mode of the fabric";
```

```
      }

      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 fabrictypes:service-capabilities;
        description
          "Supported services of the fabric";
      }
    }

    grouping device-attributes {
      description "device attributes";
      leaf device-ref {
        type fabrictypes:node-ref;
        description "The device the fabric includes.";
      }
      leaf-list role {
        type fabrictypes:device-role;
        default fabrictypes:leaf;
        description "Role of the node";
      }
    }

    grouping link-attributes {
      description "Link attributes";
      leaf link-ref {
        type fabrictypes:link-ref;
        description "The link it includes";
      }
    }

    grouping port-attributes {
      description "Port attributes";
      leaf port-ref {
        type fabrictypes:tp-ref;
        description "The port it refers to.";
```

```
      }
      leaf port-type {
        type fabrictypes:port-type;
        description
          "Port type: ethernet or serial or others";
        }
      leaf bandwidth {
        type fabrictypes:bandwidth;
        description "Bandwidth of the port";
      }
    }

    grouping fabric-attributes {
      description "Attributes of a fabric";
      leaf fabric-id {
        type fabric:fabric-id;
        description "Fabric id";
      }
      leaf name {
        type string;
        description     "Name of the fabric";
      }
      leaf type {
        type fabrictypes:underlay-network-type;
         description
          "The type of physical network that implements this
          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 "Device nodes that include in a fabric.";
        uses device-attributes;
      }
      list device-links {
        key link-ref;
        description "Links that are included within the fabric.";
        uses link-attributes;
      }
      list device-ports {
        key port-ref;
        description "Ports that are included within the  fabric.";
        uses port-attributes;
      }
    }

    // augment statements

    augment "/nws:networks/nws:network/nws:network-types" {
      description
        "Introduce new network type for Fabric-based logical topology";
      uses fabric-network-type;
    }

    augment "/nws:networks/nws:network/nws:node" {
      when "/nws:networks/nws:network/nws:network-types/sfabric:fabric-network"
{
        description "Augmentation parameters apply only for networks with
        fabric topology.";
      }
      description "Augmentation for fabric nodes.";
      container fabric-attributes-state {
        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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

J. Dong
X. Wei
Huawei Technologies

March 5, 2018

A YANG Data Model for Layer-2 Network Topologies
draft-ietf-i2rs-yang-l2-network-topology-04

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 https://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 September 6, 2018.

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-i2rs-yang-network-topo] defines the YANG [RFC6020] [RFC7950] 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 to 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:

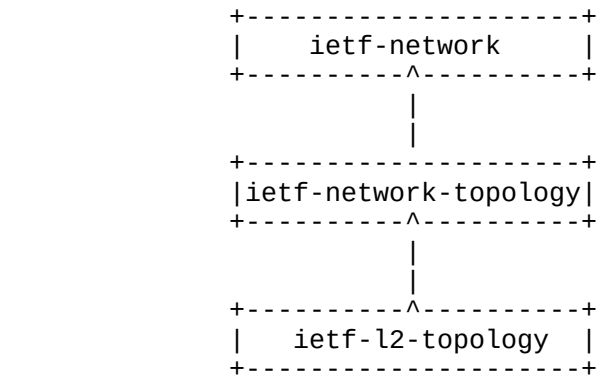


Figure 1. L2-topology model structure

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. The notation syntax follows [I-D.ietf-netmod-yang-tree-diagrams]. For purpose of brevity, notifications are not depicted.

```
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?
      |  +--ro node-ref?
      |  +--ro network-ref?
      |  +--ro l2-network!
      |  +--ro l2-node-attributes
      +---n l2-link-event
      |  +--ro event-type?
      |  +--ro link-ref?
      |  +--ro network-ref?
      |  +--ro l2-network!
      |  +--ro l2-link-attributes
      +---n l2-termination-point-event
         +--ro event-type?
         +--ro tp-ref?
         +--ro node-ref?
         +--ro network-ref?
         +--ro l2-network!
         +--ro l2-termination-point-attributes
```

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@2018-03-05.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";
      reference "RFC 6991";
    }

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

```
      reference "RFC 6991";
    }

    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>
       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.

       Copyright (c) 2018 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-ietf-i2rs-yang-l2-network-topo-04;
       see the RFC itself for full legal notices.";

    revision "2018-03-05" {
      description "Initial revision";
      reference "draft-ietf-i2rs-l2-network-topology-04";
    }

   /*
    * 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

   /*
    * 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/l2t: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/l2t: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/l2t: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/l2t: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 nw: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 nt: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 nt:tp-ref;
        uses l2-network-type;
        uses l2-termination-point-attributes;
      }
```

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

4. IANA Considerations

This document registers the following namespace URIs in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-l2-topology
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-l2-topology-state
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG modules in the "YANG Module Names" registry [RFC6020]:

Name: ietf-l2-topology
Namespace: urn:ietf:params:xml:ns:yang:ietf-l2-topology
Prefix: l2t
Reference: draft-ietf-i2rs-yang-l2-network-topology-04.txt (RFC form)

Name: ietf-l2-topology-state
Namespace: urn:ietf:params:xml:ns:yang:ietf-l2-topology-state
Prefix: l2t-s
Reference: draft-ietf-i2rs-yang-l2-network-topology-04.txt (RFC form)

5. Security Considerations

The YANG module defined in this document is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040] . The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

In general, Layer 2 network topologies are system-controlled and provide ephemeral topology information. In an NMDA-complient server, they are only part of <operational> which provides read-only access to clients, they are less vulnerable. That said, the YANG module does in principle allow information to be configurable.

The Layer 2 topology module define information that can be configurable in certain instances, for example in the case of virtual topologies that can be created by client applications. In such cases, a malicious client could introduce topologies that are undesired. Specifically, a malicious client could attempt to remove or add a node, a link, a termination point, by creating or deleting corresponding elements in the node, link, and termination point lists, respectively. In the case of a topology that is learned, the server will automatically prohibit such misconfiguration attempts. In the case of a topology that is configured, i.e. whose origin is "intended", the undesired configuration could become effective and be reflected in the operational state datastore, leading to disruption of services provided via this topology might be disrupted. For those reasons, it is important that the NETCONF access control model is vigorously applied to prevent topology misconfiguration by unauthorized clients.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability in the ietf-network module:

l2-network-attributes: A malicious client could attempt to sabotage the configuration of any of the contained attributes, such as the name or the flag data nodes.

l2-node-attributes: A malicious client could attempt to sabotage the configuration of important node attributes, such as the name or the management-address.

l2-link-attributes: A malicious client could attempt to sabotage the configuration of important link attributes, such as the rate or the delay data nodes.

l2-termination-point-attributes: A malicious client could attempt to sabotage the configuration of important termination point attributes, such as the maximum-frame-size.

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., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A Data Model for Network Topologies", draft-ietf-i2rs-yang-network-topo-20 (work in progress), December 2017.

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

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

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

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

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

7.2. Informative References

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-10 (work in progress), January 2018.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-06 (work in progress), February 2018.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Appendix A. Companion YANG model for non-NMDA compliant implementations

The YANG module ietf-l2-topology defined in this document augments two modules, ietf-network and ietf-network-topology, that are designed to be used in conjunction with implementations that support the Network Management Datastore Architecture (NMDA) defined in [I-D.ietf-netmod-revised-datastores]. In order to allow implementations to use the model even in cases when NMDA is not supported, a set of companion modules have been defined that represent a state model of networks and network topologies, ietf-network-state and ietf-network-topology-state, respectively.

In order to be able to use the model for layer 2 topologies defined in this document in conjunction with non-NMDA compliant implementations, a corresponding companion module is defined that represent the operational state of layer 2 network topologies. The module ietf-l2-topology-state mirrors the module ietf-l2-topology defined earlier in this document. However, it augments ietf-network-state and ietf-network-topology-state (instead of ietf-network and ietf-network-topology) and all its data nodes are non-configurable.

The companion module ietf-l2-topology SHOULD NOT be supported by implementations that support NMDA. It is for this reason that this module is defined in the Appendix.

As the structure of this modules mirrors that of its underlying modules, the YANG tree is not depicted separately.

```
<CODE BEGINS> file "ietf-l2-topology-state@2018-03-05.yang"
module ietf-l2-topology-state {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-l2-topology-state";
    prefix "l2t-s";

    import ietf-network-state {
      prefix "nw-s";
    }

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

    import ietf-l2-topology {
      prefix "l2t";
    }

    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>
       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.

       Copyright (c) 2018 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-ietf-i2rs-yang-l2-network-topo-04;
       see the RFC itself for full legal notices.";
```

```
       revision "2018-03-05" {
         description "Initial revision";
         reference "draft-ietf-i2rs-l2-network-topology-04";
       }



       /*
       * 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";
```

```
       }


      /*
       * Data nodes
       */


       augment "/nw-s:networks/nw-s:network/nw-s:network-types" {
         description
           "Introduce new network type for L2 topology";
         uses l2t:l2-network-type;
       }

       augment "/nw-s:networks/nw-s:network" {
         when "/nw-s:networks/nw-s:network/nw-s:network-types/"+
           "l2t-s:l2-network" {
           description
             "Augmentation parameters apply only for networks
              with L2 topology";
         }
         description
           "Configuration parameters for the L2 network
            as a whole";
         uses l2t:l2-network-attributes;
       }

       augment "/nw-s:networks/nw-s:network/nw-s:node" {
         when "/nw-s:networks/nw-s:network/nw-s:network-types/"+
           "l2t-s:l2-network" {
           description
             "Augmentation parameters apply only for networks
              with L2 topology";
         }
         description
           "Configuration parameters for L2 at the node
            level";
         uses l2t:l2-node-attributes;
       }

       augment "/nw-s:networks/nw-s:network/nt-s:link" {
         when "/nw-s:networks/nw-s:network/nw-s:network-types/"+
           "l2t-s:l2-network" {
           description
             "Augmentation parameters apply only for networks
              with L2 topology";
         }
```

```
        description "Augment L2 topology link information";
        uses l2t:l2-link-attributes;
      }

      augment "/nw-s:networks/nw-s:network/nw-s:node/"+
        "nt-s:termination-point" {
        when "/nw-s:networks/nw-s:network/nw-s:network-types/"+
          "l2t-s:l2-network" {
          description
            "Augmentation parameters apply only for networks
             with L2 topology";
        }
        description
          "Augment L2 topology termination point information";
        uses l2t:l2-termination-point-attributes;
      }

     /*
      * Notifications
      */

      notification l2-node-event {
        description "Notification event for L2 node";
        leaf event-type {
          type l2t:l2-network-event-type;
          description "Event type";
        }
        uses nw-s:node-ref;
        uses l2t:l2-network-type;
        uses l2t:l2-node-attributes;
      }

      notification l2-link-event {
        description "Notification event for L2 link";
        leaf event-type {
          type l2t:l2-network-event-type;
          description "Event type";
        }
        uses nt-s:link-ref;
        uses l2t:l2-network-type;
        uses l2t:l2-link-attributes;
      }

      notification l2-termination-point-event {
        description "Notification event for L2 termination point";
        leaf event-type {
          type l2t:l2-network-event-type;
          description "Event type";
```

```
        }
        uses nt-s:tp-ref;
        uses l2t:l2-network-type;
        uses l2t:l2-termination-point-attributes;
      }

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

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