Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2019

A. Lindem
Cisco Systems
Y. Qu
Huawei

March 4, 2019

# RIB YANG Data Model
## draft-acee-rtgwg-yang-rib-extend-10.txt

Abstract

The Routing Information Base (RIB) is a list of routes and their corresponding administrative data and operational state.

RFC 8349 defines the basic building blocks for RIB, and this model augments it to support multiple next-hops (aka, paths) for each route as well as additional attributes.

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 5, 2019.

Table of Contents



## 1. Introduction

This document defines a YANG, [RFC6020][RFC7950], data model which extends the generic data model for RIB by augmenting the ietf-routing model as defined in [RFC8349].

RIB is a collection of best routes from all routing protocols. Within a protocol routes are selected based on the metrics in use by that protocol, and the protocol install its best routes to RIB. RIB selects the best route by comparing the route preference (aka, administrative distance) of the associated protocol.

The augmentations described herein extend the RIB to support multiple paths per route, route metrics, and administrative tags.

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

## 2. Terminology and Notation

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

The following terms are defined in [RFC8342]:

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

The following terms are defined in [RFC7950]:

- action
- augment
- container
- container with presence
- data model
- data node
- feature
- leaf
- list
- mandatory node
- module
- schema tree

- RPC (Remote Procedure Call) operation

### 2.1. Glossary of New Terms

Routing Information Base (RIB): An object containing a list of routes, together with other information. See [RFC8349] Section 5.2 for details.

### 2.2. Tree Diagrams

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

### 2.3. Prefixes in Data Node Names

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

| Prefix | YANG module | Reference |
|---|---|---|
| if | ietf-interfaces | [RFC8343] |
| rt | ietf-routing | [RFC8349] |
| v4ur | ietf-ipv4-unicast-routing | [RFC8349] |
| v6ur | ietf-ipv6-unicast-routing | [RFC8349] |
| inet | ietf-inet-types | [RFC6991] |

Table 1: Prefixes and Corresponding YANG Modules

## 3. Design of the Model

The YANG definitions in this document augment the ietf-routing model defined in [RFC8349], which provides a basis for routing system data model development. Together with modules defined in [RFC8349], a generic RIB Yang model is defined to implement and monitor RIB.

The models in [RFC8349] also define the basic configuration and operational state for both IPv4 and IPv6 static routes and this document also provides augmentations for static routes to support multiple next-hop and more next-hop attributes.

### 3.1. RIB Tags and Preference

Individual routes tags will be supported at both the route and next-hop level. A preference per next-hop is also supported for selection of the most preferred reachable static route.

### 3.2. Multiple next-hops

Both Ipv4 and IPv6 static route configuration defined in [RFC8349] have been augmented with a multi-next-hop option.

A static route/prefix can be configured to have multiple next-hops, each with their own tag and route preference.

In RIB, a route may have multiple next-hops. They can be either equal cost multiple paths (ECMP), or they may have different metrics.

### 3.3. Repair path

The loop-free alternate (LFA) Fast Reroute (FRR) pre-computes repair paths by routing protocols, and RIB stores the best repair path.

A repair path is augmented in RIB operation state for each path.

## 4. RIB Model Tree

The tree associated with the "ietf-rib-extension" module follows. The meaning of the symbols can be found in [RFC8340]. The ietf-routing.yang tree with the augmentations herein is included in Appendix A.

```
augment /rt:routing/rt:control-plane-protocols
         /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
         /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
         /v4ur:simple-next-hop:
 +--rw preference?        uint32
 +--rw tag?               uint32
 +--rw application-tag?   uint32
augment /rt:routing/rt:control-plane-protocols
         /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
         /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
         /v4ur:next-hop-list/v4ur:next-hop-list/v4ur:next-hop:
 +--rw preference?        uint32
 +--rw tag?               uint32
 +--rw application-tag?   uint32
augment /rt:routing/rt:control-plane-protocols
         /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
         /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
```

```
          /v6ur:simple-next-hop:
    +--rw preference?        uint32
    +--rw tag?               uint32
    +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
            /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
            /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
            /v6ur:next-hop-list/v6ur:next-hop-list/v6ur:next-hop:
    +--rw preference?        uint32
    +--rw tag?               uint32
    +--rw application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib:
    +--ro rib-summary-statistics
       +--ro total-routes?              uint32
       +--ro total-active-routes?       uint32
       +--ro total-route-memory?        uint64
       +--ro protocol-rib-statistics* []
          +--ro rib-protocol?             identityref
          +--ro protocol-total-routes?    uint32
          +--ro protocol-active-routes?   uint32
          +--ro protocol-route-memory?    uint64
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route:
    +--ro metric?            uint32
    +--ro tag?               uint32
    +--ro application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes:
    +--ro repair-route* [id]
       +--ro id          string
       +--ro next-hop
       |  +--ro outgoing-interface?   if:interface-state-ref
       |  +--ro next-hop-address?     inet:ip-address
       +--ro metric?     uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route
            /rt:next-hop/rt:next-hop-options/rt:simple-next-hop:
    +--ro repair-path?
            -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route
            /rt:next-hop/rt:next-hop-options/rt:special-next-hop:
    +--ro repair-path?
            -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route
            /rt:next-hop/rt:next-hop-options/rt:next-hop-list
            /rt:next-hop-list/rt:next-hop:
    +--ro repair-path?
            -> /rt:routing/ribs/rib/routes/repair-route/id
```

## 5. RIB YANG Model

```
<CODE BEGINS> file "ietf-rib-extension@2019-03-01.yang"
module ietf-rib-extension {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-rib-extension";

  prefix rib-ext;

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

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-routing {
    prefix "rt";
  }

  import ietf-ipv4-unicast-routing {
    prefix "v4ur";
  }

  import ietf-ipv6-unicast-routing {
    prefix "v6ur";
  }

  organization
      "IETF RTGWG - Routing Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/group/rtgwg/>
     WG List:  <mailto:rtgwg@ietf.org>

     Author:   Acee Lindem
               <mailto:acee@cisco.com>
     Author:   Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>";

  description
    "This YANG module extends the generic data model for
     RIB by augmenting the ietf-netmod-routing-cfg
     model. It is intended that the module will be extended
     by vendors to define vendor-specific RIB parameters.

     This YANG model conforms to the Network Management
```

```
        Datastore Architecture (NDMA) as described in RFC 8342.

        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 RFC XXXX;
        see the RFC itself for full legal notices.";

    revision 2019-03-01 {
      description
        "Initial RFC Version";
      reference
        "RFC XXXX: A YANG Data Model for RIB Extensions.";
    }

    /* Groupings */
    grouping rib-statistics {
      description "Statistics grouping used for RIB augmentation";
      container rib-summary-statistics {
        config false;
        description "Container for RIB statistics";
        leaf total-routes {
          type uint32;
          description
            "Total routes in the RIB from all protocols";
        }
        leaf total-active-routes {
          type uint32;
          description
            "Total active routes in the RIB";
        }
        leaf total-route-memory {
          type uint64;
          description
            "Total memory for all routes in the RIB from all
             protocol clients";
        }
        list protocol-rib-statistics {
          description "List protocol statistics";
          leaf rib-protocol {
            type identityref {
```

```
            base rt:routing-protocol;
          }
          description "Routing protocol for statistics";
        }
        leaf protocol-total-routes {
          type uint32;
          description
            "Total number routes for protocol in the RIB";
        }
        leaf protocol-active-routes {
          type uint32;
          description
            "Number active routes for protocol in the RIB";
        }
        leaf protocol-route-memory {
          type uint64;
          description
            "Total memory for all routes in the RIB for protocol";
        }
      }
    }
  }

  grouping next-hop {
    description
      "Next-hop grouping";
    leaf interface {
      type if:interface-ref;
      description
        "Outgoing interface";
    }
    leaf address {
      type inet:ip-address;
      description
        "IPv4 or IPv6 Address of the next-hop";
    }
  }

  grouping attributes {
    description
      "Common attributes applicable to all paths";
    leaf metric {
      type uint32;
      description "Route metric";
    }
    leaf tag {
      type uint32;
      description "Route tag";
```

```
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
    }
    grouping path-attribute {
      description
        "Path attribute grouping";
      leaf repair-path {
        type leafref {
          path "/rt:routing/rt:ribs/rt:rib/"
               + "rt:routes/repair-route/id";
        }
        description
          "IP Fast ReRoute (IPFRR) repair path, use a path
           from repair-route list";
      }
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v4ur:ipv4/"
          + "v4ur:route/v4ur:next-hop/v4ur:next-hop-options/"
          + "v4ur:simple-next-hop"
    {
      description
        "Augment 'simple-next-hop' case in IPv4 unicast route.";
      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v4ur:ipv4/"
```

```
          + "v4ur:route/v4ur:next-hop/v4ur:next-hop-options/"
          + "v4ur:next-hop-list/v4ur:next-hop-list/v4ur:next-hop"
    {
      description
        "Augment static route configuration 'next-hop-list'.";

      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
  }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v6ur:ipv6/"
          + "v6ur:route/v6ur:next-hop/v6ur:next-hop-options/"
          + "v6ur:simple-next-hop"
    {
      description
        "Augment 'simple-next-hop' case in IPv6 unicast route.";
      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
```

```
      }
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes/v6ur:ipv6/"
          + "v6ur:route/v6ur:next-hop/v6ur:next-hop-options/"
          + "v6ur:next-hop-list/v6ur:next-hop-list/v6ur:next-hop"
    {
      description
        "Augment static route configuration 'next-hop-list'.";

      leaf preference {
        type uint32;
        default "1";
        description "Route preference - Used to select among multiple
                     static routes with a lower preference next-hop
                     preferred and equal preference paths yielding
                     Equal Cost Multi-Path (ECMP).";
      }
      leaf tag {
        type uint32;
        default "0";
        description "Route tag";
      }
      leaf application-tag {
        type uint32;
        description "Additional Application-Specific Route tag";
      }
    }

    augment "/rt:routing/rt:ribs/rt:rib"
    {
      description "Augment a RIB with statistics";
      uses rib-statistics;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route"
    {
      description
        "Augment a route in RIB with tag.";
      uses attributes;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes"
    {
      description
```

```
        "Augment a route with a list of repair-paths.";
      list repair-route {
        key "id";
        description
          "A repair-path entry, which can be referenced
           by a repair-path.";
        leaf id {
          type string;
          description
            "A unique identifier.";
        }

        container next-hop {
                description
                  "Route's next-hop attribute.";
          leaf outgoing-interface {
            type if:interface-state-ref;
            description
              "Name of the outgoing interface.";
          }
          leaf next-hop-address {
            type inet:ip-address;
            description
              "IP address of the next hop.";
          }
        }
        leaf metric {
          type uint32;
          description "Route metric";
        }
      }
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:simple-next-hop"
    {
       description
         "Add more parameters to a path.";
       uses path-attribute;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:special-next-hop"
    {
      description
        "Add more parameters to a path.";
```

```
      uses path-attribute;
    }

    augment "/rt:routing/rt:ribs/rt:rib/"
          + "rt:routes/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:next-hop-list/rt:next-hop-list/rt:next-hop"
    {
      description
        "This case augments the 'next-hop-options' in the routing
         model.";
      uses path-attribute;
    }
  }
  <CODE ENDS>
```

## 6. Security Considerations

The YANG modules specified in this document define 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 [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a pre-configured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in ietf-rib-extensions.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. For these augmentations to ietf-routing.yang, the abiliy to delete, add, and modify IPv4 and IPv6 static routes would allow traffic to be misrouted.

Some of the readable data nodes in the ietf-rib-extensions.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 exposure of the Routing Information Base (RIB) will expose the routing topology of the network. This may be undesirable since both due to the fact that exposure may facilitate other attacks. Additionally, network operators may consider their topologies to be sensitive confidential data.

All the security considerations for [RFC8349] writable and readable data nodes apply to the augmentations described herein.

## 7. IANA Considerations

This document registers a URI in the IETF XML registry [XML-REGISTRY]. Following the format in [RFC3688], the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-rib-extension

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-acl namespace: urn:ietf:params:xml:ns:yang:ietf-rib-extension prefix: ietf-rib-ext reference: RFC XXXX

## 8. References

### 8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <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>.

[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>.

[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>.

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

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

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

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

## 8.2. Informative References

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

[XML-REGISTRY] Mealling, M., "The IETF XML Registry", BCP 81, January 2004.

## Appendix A. Combined Tree Diagram

This appendix includes the combined ietf-routing.yang and ietf-rib-extensions.yang tree diagram.

```
module: ietf-routing
+--rw routing
|  +--rw router-id?                 yang:dotted-quad
|  +--ro interfaces
|  |  +--ro interface*   if:interface-ref
|  +--rw control-plane-protocols
|  |  +--rw control-plane-protocol* [type name]
|  |     +--rw type             identityref
|  |     +--rw name             string
|  |     +--rw description?     string
|  |     +--rw static-routes
|  +--rw ribs
|     +--rw rib* [name]
|        +--rw name               string
|        +--rw address-family    identityref
|        +--ro default-rib?      boolean {multiple-ribs}?
|        +--ro routes
|        |  +--ro route* []
|        |     +--ro route-preference?   route-preference
|        |     +--ro next-hop
|        |     |  +--ro (next-hop-options)
|        |     |     +--:(simple-next-hop)
|        |     |     |  +--ro outgoing-interface?   if:interface-ref
|        |     |     +--:(special-next-hop)
|        |     |     |  +--ro special-next-hop?     enumeration
|        |     |     +--:(next-hop-list)
|        |     |        +--ro next-hop-list
|        |     |           +--ro next-hop* []
|        |     |              +--ro outgoing-interface?
|        |     |                      if:interface-ref
|        |     +--ro source-protocol     identityref
|        |     +--ro active?             empty
|        |     +--ro last-updated?       yang:date-and-time
|        +---x active-route
|        |  +--ro output
|        |     +--ro route
|        |        +--ro next-hop
|        |        |  +--ro (next-hop-options)
|        |        |     +--:(simple-next-hop)
|        |        |     |  +--ro outgoing-interface?
|        |        |     |          if:interface-ref
|        |        |     +--:(special-next-hop)
|        |        |     |  +--ro special-next-hop?     enumeration
```

```
|         |         |     +--:(next-hop-list)
|         |         |        +--ro next-hop-list
|         |         |           +--ro next-hop* []
|         |         |              +--ro outgoing-interface?
|         |         |                          if:interface-ref
|         |         +--ro source-protocol    identityref
|         |         +--ro active?            empty
|         |         +--ro last-updated?      yang:date-and-time
|         +--rw description?      string
o--ro routing-state
   +--ro router-id?                  yang:dotted-quad
   o--ro interfaces
   |  o--ro interface*   if:interface-state-ref
   o--ro control-plane-protocols
   |  o--ro control-plane-protocol* [type name]
   |     o--ro type    identityref
   |     o--ro name    string
   o--ro ribs
      o--ro rib* [name]
         o--ro name               string
         +--ro address-family     identityref
         o--ro default-rib?       boolean {multiple-ribs}?
         o--ro routes
         |  o--ro route* []
         |     o--ro route-preference?   route-preference
         |     o--ro next-hop
         |     |  +--ro (next-hop-options)
         |     |     +--:(simple-next-hop)
         |     |     |  +--ro outgoing-interface?   if:interface-ref
         |     |     +--:(special-next-hop)
         |     |     |  +--ro special-next-hop?     enumeration
         |     |     +--:(next-hop-list)
         |     |        +--ro next-hop-list
         |     |           +--ro next-hop* []
         |     |              +--ro outgoing-interface?
         |     |                          if:interface-ref
         |     +--ro source-protocol     identityref
         |     +--ro active?             empty
         |     +--ro last-updated?       yang:date-and-time
         o---x active-route
            +--ro output
               o--ro route
                  o--ro next-hop
                  |  +--ro (next-hop-options)
                  |     +--:(simple-next-hop)
                  |     |  +--ro outgoing-interface?
                  |     |            if:interface-ref
                  |     +--:(special-next-hop)
```

```
                     |      |  +--ro special-next-hop?     enumeration
                     |      +--:(next-hop-list)
                     |         +--ro next-hop-list
                     |            +--ro next-hop* []
                     |               +--ro outgoing-interface?
                     |                       if:interface-ref
                     +--ro source-protocol    identityref
                     +--ro active?            empty
                     +--ro last-updated?      yang:date-and-time

   module: ietf-rib-extension
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
             /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
             /v4ur:simple-next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v4ur:ipv4
             /v4ur:route/v4ur:next-hop/v4ur:next-hop-options
             /v4ur:next-hop-list/v4ur:next-hop-list/v4ur:next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
             /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
             /v6ur:simple-next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:control-plane-protocols
             /rt:control-plane-protocol/rt:static-routes/v6ur:ipv6
             /v6ur:route/v6ur:next-hop/v6ur:next-hop-options
             /v6ur:next-hop-list/v6ur:next-hop-list/v6ur:next-hop:
     +--rw preference?        uint32
     +--rw tag?               uint32
     +--rw application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib:
     +--ro rib-summary-statistics
        +--ro total-routes?              uint32
        +--ro total-active-routes?       uint32
        +--ro total-route-memory?        uint64
        +--ro protocol-rib-statistics* []
           +--ro rib-protocol?             identityref
           +--ro protocol-total-routes?    uint32
           +--ro protocol-active-routes?   uint32
```

```
         +--ro protocol-route-memory?    uint64
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route:
     +--ro metric?            uint32
     +--ro tag?               uint32
     +--ro application-tag?   uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes:
     +--ro repair-route* [id]
        +--ro id          string
        +--ro next-hop
        |  +--ro outgoing-interface?   if:interface-state-ref
        |  +--ro next-hop-address?     inet:ip-address
        +--ro metric?     uint32
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route/rt:next-hop
             /rt:next-hop-options/rt:simple-next-hop:
     +--ro repair-path?   -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route/rt:next-hop
             /rt:next-hop-options/rt:special-next-hop:
     +--ro repair-path?   -> /rt:routing/ribs/rib/routes/repair-route/id
   augment /rt:routing/rt:ribs/rt:rib/rt:routes/rt:route/rt:next-hop
             /rt:next-hop-options/rt:next-hop-list/rt:next-hop-list
             /rt:next-hop:
     +--ro repair-path?   -> /rt:routing/ribs/rib/routes/repair-route/id
```

## Appendix B. ietf-rib-extension.yang examples

Examples will be added in a future version of this document.

## Appendix C. Acknowledgments

The RFC text was produced using Marshall Rose's xml2rfc tool.

The authors wish to thank Les Ginsberg, Krishna Deevi, and Suyoung Yoon for their helpful comments and suggestions.

The authors wish to thank Tom Petch and Rob Wilton for review and comments.

## Authors' Addresses

Acee Lindem
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

EMail: acee@cisco.com

Yingzhen Qu
Huawei
2330 Central Expressway
Santa Clara, CA 95050
USA

EMail: yingzhen.qu@huawei.com

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: October 6, 2016

S. Hares
Huawei
R. White
LinkedIn

April 4, 2016

# Filter-Based RIB Data Model
# draft-hares-rtgwg-fb-rib-01

## Abstract

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

## Status of This Memo

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

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

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

This Internet-Draft will expire on October 6, 2016.

## Copyright Notice

Table of Contents



## 1. Introduction

This document provides a yang module for flow filter n-tuple policy that is locally configured. This flow filter policy has also been called Policy routing in some implementations.

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

### 1.1. Definition of I2RS Filter Based RIB

Filter-based routing is a technique used to make packet forwarding decisions based on a n-tuple filter that is matched to the incoming packets and the specified action. It should be noted that that this is distinct from the static routes in the following RIBS:

- configured RIB created using static routes in [I-D.ietf-netmod-routing-cfg]

- o Extended static RIB defined in [I-D.acee-rtgwg-yang-rib-extend],

- o Ephmeral Protocol Independent RIB defined in [I-D.ietf-i2rs-rib-info-model], or

A Filter-Based RIB (Routing Information Base) is contained in a routing instance. It contains a list of filters (match-action conditions), a list of interface the filter-based forwarding operates on. Filter-based RIBs (FB-RIBs) operate only on the interface the FB-RIB are configured on.

A Filter Based RIB uses packet forwarding policy. If packet reception is considered an event, then the I2RS Filter-based RIB uses a minimalistic Event-Condition-Action policy. A Filter-based RIB entry specifies matche filters for the fields in a packet (which may include layer 1 to layer 3 header fields, transport or application fields) or size of the packet or interface received on. The matches are contained in an ordered list of filters which contain pairs of match condition-action (aka event-condition-action).

If all matches fail, the default action is to forward the packet using FIB entries that were programmed by the default Routing Informational Base (RIB) manager configured in the Filter-Based RIB (FB-RB)

Actions in the condition-action pair may impact forwarding or set something in the packet that will impact forwarding. Policy actions are typically applied before applying QoS constraints since policy actions may override QoS constraint.

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

## 1.2. Requirements Language

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

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

## 1.3. Definitions and Acronyms

CLI

Command Line Interface

FB-RIB

Filter-Based Routing Information Base

FB-Route

The policy rules in the filter-based RIB are prescriptive of the Event-Condition-Action form which is often represented by if Condition then action. All policy in the filter-based RIB are in a ordered list, ordered by "order-number". Order number is similar to some CLI concepts of line number.

Policy Group

Policy Groups are groups of policy rules that are set-up for the convenience of operators who wish to link the rules connected to a particular client.

* Groups do not affect the order of policy rulies.

* The policy groups in the basic network policy [I-D.hares-i2rs-pkt-eca-data-model] allow grouping of policy by name. This name allow easier management of customer-based or provider based filters. This policy group is a second way to access certain policy rules on the policy rule list.

RIB IM

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

Routing instance

A routing instance, in the context of the FB-FIB is a collection of RIBs, interfaces, and routing parameters. A routing instance creates a logical slice of the router and allows different logical slices; across a set of routers; to communicate with each other.

## 1.4. Yang High Level (YHL) graphical form

The High-level Yang graphical representation uses the following symbols:

Brackets "[" and "]" enclose list keys.

Curly braces "{" and "}" contain names of optional features that make the corresponding node conditional.

Abbreviations before data node names: "rw" means configuration (read-write), "ro" state data (read-only), "-x" RPC operations, and "-n" notifications.

Symbols after data node names: "?" means an optional node, "!" a container with presence, and "*" denotes a "list" or "leaf-list".

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

## 2. Where Filter-Based RIB Fits in Global RIBs

The Top-level Yang structure for a global FB-RIB types (similar to acl) is not defined. The Filter-Based RIB should be defined under this structure under a routing instance. The two things under this RIB would be: configured Filter-Based RIB (aka Policy routing), I2RS reboot Ephemeral Filter-Based RIB. ACLs [I-D.ietf-netmod-acl-model] have the potential to be augmented to be included, but this version of this document does address that issue.

The purpose of this section is illustrate why the flow specification policy installed in yang modules loaded into intended configuration needs to be able to be compared. After demonstrating why this is needed, this section suggests a structure for filter-based RIBS.

BGP’s Flow Specification (BGP-FS) configures filter-based policy in the local BGP configuration, and passes this information in BGP packets (in NLRI and Extended Communities). The BGP-FS YANG model [I-D.wu-idr-flowspec-yang-cfg] specifies the locally configuration, and the derived state that includes the BGP Flow Specifications received. BGP-FS processing may install the locally configured BGP Flow specification in the local FB-RIB. If it does, this policy is like any other locally configured policy.

The BGP-FS may installed the flow policy received from a remote BGP peer and stored in derived state. This policy has a different characteristics as it will disappear if the peer connection between the two peers drops, or if the peer changes the BGP-FS policy. Due to the ephemeral nature of the BGP-FS, it should be installed unique. Otherwise, If the local configuration state changes, it cannot differentiate between the true configured state and the ephemeral states (I2RS ephemeral and BGP-session ephemeral). Both I2RS ephemeral and BGP-session ephemeral policy will disappear upon a reboot.

```
ietf-fb-rib module
+--rw routing-instance
   +--rw ietf-fb-rib
      +--rw default-instance-name string
      +--rw default-router-id rt:router-id
          +--rw config-fb-rib // config state
             uses fb-ribs
          +--rw I2rs-fb-rib  // ephemeral state
             uses fb-ribs
          +--rw BGP-FB-RIB   // Install derived
             uses fb-ribs    // BGP-FS policy state
```

Figure 6: Global FB RIB Yang Structure

I2RS architecture [I-D.ietf-i2rs-architecture] specifies that by default the Local configuration will win if the local configuration changes. In the NETCONF/NETMOD language, the "last write wins".

An example will help illustrate this:

local configuration installs filter for IP-Dest=128.2/16, IP-SRC=192.5.7/24 DPORT=ALL drop in the running configuration, and then synchronously loads it to the intended configuration and applied configuration.

I2RS installs an ephemeral filter for IP-Dest=128.2/16, IP-SRC=192.5.7/24 DPORT=125 forward intended configuration synchronously.

BGP-FS processing installs BGP-FS policy for IP-Dest=128.2/16, IP-SRC=192.5.7/24 DPORT=125 forward, traffic-rate by bytes.

local configuration install a filter for IP-Dest=128.2/16, IP-SRC=192.5.7/24, DPort=125, drop. This local configuration policy would win over the I2RS policy and the BGP-FS. The I2RS process is required to receive an event indicating the overwrite. The BGP-FS process should also receive an event indicating an overwrite.

The I2RS [I-D.ietf-i2rs-architecture] also allows that the preference between local-configuration and I2RS ephemeral state can be determined by operator-applied policy. However, illustrations of this are out of scope for this version of this document.

## 3. Proposed Structure for Filter-Based RIBs

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

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

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

This yang model describes three types of FB-RIBS: configuration, I2RS, and BGP Flow Specification. The configuration FB-RIB yang module is config state ("config true" and "ephemeral false") and survives a reboot. The I2RS FB-RB yang model is reboot ephemeral ("config true" and "ephemeral true"). The BGP Flow Specification Filter-Based RIB stores policy which is received by the BGP peers, and can be considered policy configured as part of BGP infrastructure ("config true" and "peer-ephemeral true;")

Configuration RIBS

bgp-fs-fb-rib - is the BGP processes installation of the BGP Flow Specification (BGP-FS) policy rules from remote peers. Locally configured BGP-FS rules are configured in the BGP peer structure.

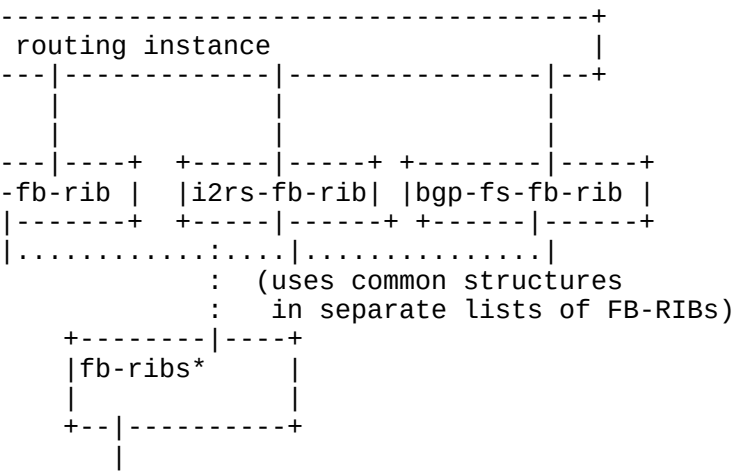



Figure 3: Routing instance with three types of Filter-FIB lists

## 4. Yang High Level Structure for FB-RIBs

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

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

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

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

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

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

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

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

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

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

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

Figure 5: configuration state

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

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

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

Figure 5: operational state

## 4.2. Filter-Based RIB structures

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

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

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

Figure 6: FB RIB Type Structure

HIgh Level Yang

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

## 5. yang models

### 5.1. Filter-Based RIB types

Yang model is contained in draft-hares-i2rs-fb-rib-data-model-01.txt

Please see this draft for the data model.

## 6. IANA Considerations

TBD

## 7. Security Considerations

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

ephemeral RIBs to alter the traffic paths set by routing configuration. Care must be taken in deployments to use the appropriate security and operational control to make use of the tools the I2RS RIB and I2RS FB-RIB provide.

## 8. References

### 8.1. Normative References:

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

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

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

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

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

[I-D.wu-idr-flowspec-yang-cfg]
Wu, N., Zhuang, S., and A. Choudhary, "A YANG Data Model for Flow Specification", draft-wu-idr-flowspec-yang-cfg-02 (work in progress), October 2015.

### 8.2. Informative References

[I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-13 (work in progress), February 2016.

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

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

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

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

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

## Authors' Addresses

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Russ White
LinkedIn

Email: russ@riw.us

INTERNET-DRAFT
Intended Status: Informational
Expires: September 22, 2016

T. Herbert
Facebook

March 21, 2016

# ILA Control Messages
# draft-herbert-ila-messages-00

## Abstract

This specification defines control messages for Identifier Locator Addressing. These control messages are sent between ILA hosts or from ILA routers to ILA hosts to notify a sending host to change its entry for an identifier in its ILA mapping table. Messages indicate that no mapping was found for a destination identifier or indicate a redirect to use a different locator for an identifier.

## Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

## Copyright and License Notice

Table of Contents

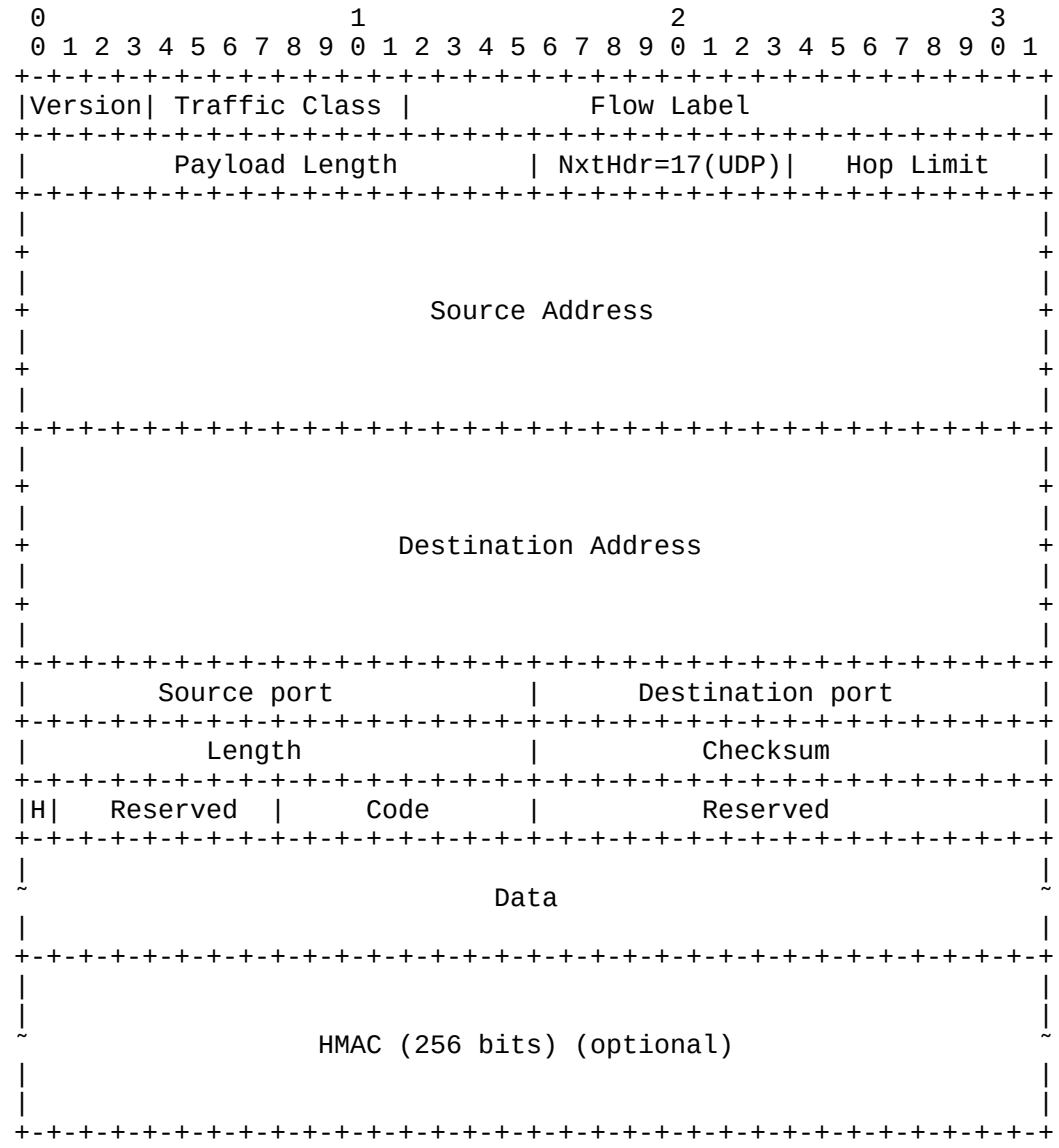
# 1 Introduction

Identifier locator addressing is described in [I-D.herbert-nvo3-ila] and an architecture for ILA deployment is described in [I-D.lapukhov-ila-deployment]. In this specification we describe ILA control messages. These are messages concerning the state of ILA mappings and are sent between ILA hosts or between ILA routers and ILA hosts. These messages are sent in response to data plane packets in order to notify a sender about status of the ILA mapping for the identifier in a destination address. There are four different types of notifications:

- "Host No Identifier" is sent by an ILA host if it receives a packet addressed with its local locator but cannot match the identifier.
- "Router No Mapping" is sent by an ILA router when it receives a SIR addressed packet however does not have a mapping for the identifier. In this case the destination identifier is unreachable
- "Host redirect" is sent by an ILA host when it receives a packet addressed with its local locator and the identifier has a forwarding mapping entry. This is used to redirect senders to a new location after an identifier has been moved to a different host.
- "Router redirect" is sent by an ILA router when it receives a SIR addressed packet and the locator for the identifier is in the mapping table. Upon receiving this message a host is able to send packets directly using the locator instead of the SIR address.

2 ILA control message format

ILA messages are sent in the payload of UDPv6 packets. The general format is:




o Source address is the common locator address (CLA) of the device

sending the message. (the common locator address has the form: <locator>::1).

- Destination address is the common locator address derived from the source address of the received packet. The source address of the received packet is a SIR address, so to send an ILA control message the SIR address must be reverse mapped to a locator which is set in the CLA.

- Source port: set to ILA control message port number (assignment TDB).

- Destination UDP port: set to ILA control message port number (assignment TBD).

- H: Indicates the HMAC field is present.

- Code indicates the type of the ILA message. Defined codes are:

  - 0x1: Host No Identifier

  - 0x2: Router No Mapping

  - 0x3: Host Redirect

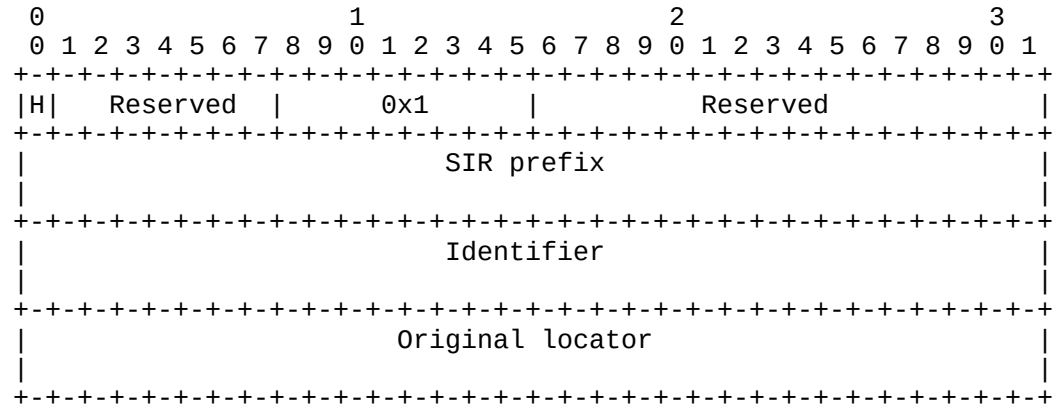
  - 0x4: Router Redirect

- Data: The data portion of the control messages.

- HMAC: HMAC Key ID and HMAC field, and their use are defined in Section 4.

## 2.1 Host No Identifier

The Host No Identifier message is sent by an ILA host if it receives a packet addressed with its local locator but cannot match the identifier. The format of this message is:



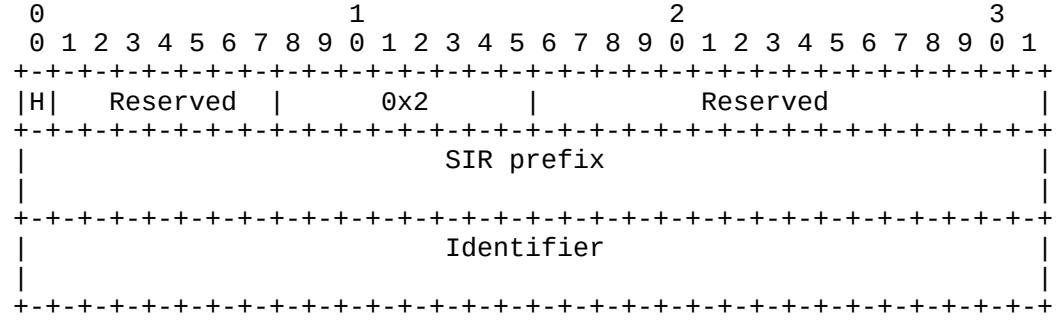

- o SIR prefix: the SIR prefix associated with the locator of the sending host (i.e. the locator in the original packet)

- o Identifier: The identifier received in the packet. The host sending the ILA message has no mapping for this identifier.

- o Original locator: The locator that was in the destination address of the received packet. If there is a only one locator assigned per host this will be the same as the locator used in the source address of the ILA message (see above).

## 2.2 Router No Mapping

Router No Mapping is sent by an ILA router when it receives a SIR addressed packet however does not have a mapping for the identifier. The format of this message is:



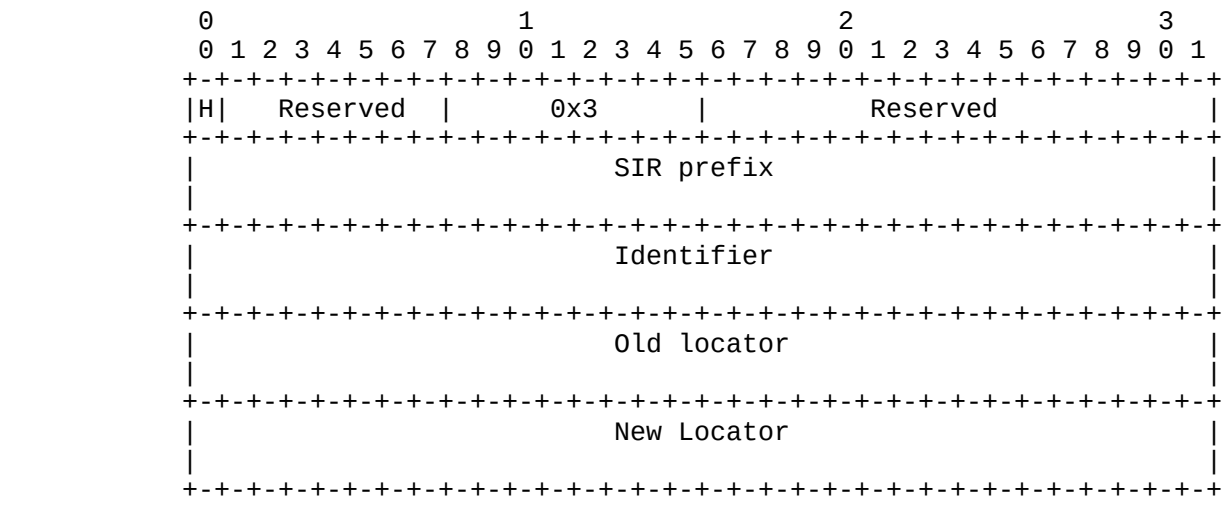

- o SIR prefix: the SIR prefix associated with the locator of the sending host (i.e. the locator in the original packet)

- o Identifier: The identifier received in the packet. The ILA

router sending the ILA message has no mapping for this identifier.

## 2.3 Host redirect

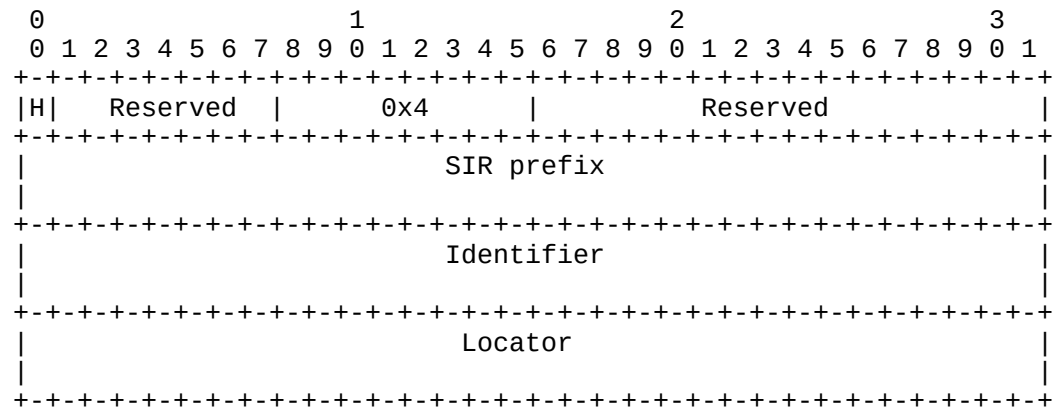
A Host Redirect is sent by an ILA host when receives a packet addressed with its local locator and the identifier has a forwarding mapping entry. The format of this message is:




- o SIR prefix: the SIR prefix associated with the locator of the sending host (i.e. the locator in the original packet)
- o Identifier: The identifier received in the packet. The host sending the ILA message is suggesting a new locator for this identifier.
- o Old locator: The locator that was set in destination of the received packet.
- o New Locator: The locator that the a sending host is being redirected to use.

## 2.4 Router redirect

A Router Redirect is sent by an ILA router when it receives a SIR addressed packet and the locator for the identifier in the destination address is in the mapping table. The format of this message is:




- o SIR prefix: the SIR prefix associated with the locator of the sending host (i.e. the locator in the original packet)

- o Identifier: The identifier received in the packet. The router sending the ILA message is suggesting a new locator for this identifier.

- o Locator: The locator that the a sending host is being redirected to use.

## 3. Operation

ILA messages can only be sent in response to packets whose source address is a SIR address in the same domain as the host or router sending the control messages. ILA messages must not be sent to non-ILA hosts or ILA hosts in a different domain (i.e. use a different SIR prefix).

ILA messages must be verified for authenticity (see Section 4).

### 3.1 Host control message generation

An ILA host may send control messages in response to packets received that have its local locator in the destination address.

The steps in receive processing of an ILA host are:

1) Receive packets where the destination locator matches the local locator of the host.

2) Lookup the identifier in the destination address in the ILA locator mapping database.

3) If the identifier is found and it is marked as a local identifier (i.e. this is the location for the identifier), then overwrite the locator in the destination address with the SIR prefix and receive the packet in the local stack as per ILA processing.

4) Else, if the source address is not a SIR address in the same ILA domain as the host, then drop the packet and take no further action.

5) Lookup the identifier in the source address in the ILA mapping table. If no entry is found or there is no locator associated with the entry, then drop the packet and take no further action.

6) If the identifier in the destination address was found in the mapping database and there is a forwarding address set in the mapping entry, then send a Host Redirect message:

   a) The destination address of the control message is set to the common locator address which is comprised of the locator associated with the identifier in the source address and ::1 in the low order 64 bits.
   b) The source address of the control message packet is set to the common locator address of the host sending the message.
   c) The SIR prefix in the control message is set to that associated with the locator the packet was received on.
   d) The identifier in the control message is set to the identifier in the destination address of the received packet.
   e) The Old Locator in the control message is set to the locator in the destination address of the received packet (should be a local locator).
   f) The New locator in the control message data is set to the corresponding value in the mapping table.

7) Else, send a Host No Identifier. The host has no information regarding the received identifier:

   a) The destination address of the control message is set to the common locator address which is comprised of the locator associated with the identifier in the source address and ::1 in the low order 64 bits.
   b) The source address of the control message packet is set to the common locator address of the host sending the message.
   c) The SIR prefix in the control message is set to that associated with the locator the packet was received on.
   d) The identifier in the control message is set to the identifier in the destination address of the received packet.

e) The Original Locator in the control message is set to the locator in the destination address of the received packet (should be a local locator).

### 3.2 ILA router processing

An ILA router should send control messages in response to packets received whose destination address is a SIR address.

The steps in receive processing of an ILA router are:

1) Receive anycast SIR addressed packets that are routed to the router.

2) Lookup the identifier in the destionation address in the ILA locator database.

3) If an entry for the identifier is found and there is an associated locator, then overwrite the SIR prefix in the destination address with the found locator and forward the packet per ILA processing.

4) Else, if the source address is not a SIR address in the same ILA domain as the host, then drop the packet and take no further action.

5) Lookup the identifier in the source address in the ILA mapping table. If no entry is found or there is no locator associated with the entry, then take no further action.

6) If the locator was found in the mapping table then send a Router Redirect message:

   a) The destination address of the control message is set to the common locator address which is comprised of the locator associated with the identifier in the source address and ::1 in the low order 64 bits.
   b) The source address of the control message packet is set to the common locator address of the ILA router sending the message.
   c) The SIR prefix in the control message is set to that associated with the locator the packet was received on.
   d) The identifier in the control message is set to the identifier in the destination address of the received packet.
   f) The Locator in the control message data is set to the corresponding value in the mapping table.

7) Else, send a Router No Mapping messages

a) The destination address of the control message is set to the common locator address which is comprised of the locator associated with the identifier in the source address and ::1 in the low order 64 bits.
b) The source address of the control message packet is set to the common locator address of the ILA router sending the message.
c) The SIR prefix in the control message is set to that associated with the locator the packet was received on.
d) The identifier in the control message is set to the identifier in the destination address of the received packet.

### 3.3 Host processing of received ILA messages

Hosts listen on the appropriate UDP port for ILA control messages. The steps in processing control messages are:

1) Control messages are received on the common locator address for the host.

2) If the message code is Host No Mapping

   a) Lookup the specified identifier in the ILA mapping cache. If an entry is found, there is a locator in the mapping, and the locator matches that in the Original Locator in the control message-- then invalidate the locator. The next packet that is sent to that identifier will not be translated and so will be sent with a destination SIR address.
   b) If the mapping entry has no locator set or a different locator than what is reported in the Host No mapping, then disregard the message

3) If the message code is Router No Mapping

   a) Lookup the specified identifier in the ILA mapping cache. If a match is found, then mark the entry is unreachable (subject to a timer)
   b) If an entry is not found then disregard the message.

4) If the message code is Host redirect

   a) Lookup the specified identifier in the ILA mapping cache. If an entry is found, there is a locator in the mapping, and the locator matches that in the Old Locator in the control message-- then set the locator to the value of New Locator. The next packet that is sent to that identifier will use the new locator.
   b) If the mapping entry has no locator set or a different

locator than what is reported in the Host Redirect then disregard the message

5) If the message is Router Redirect

a) Lookup the specified identifier in the ILA mapping cache. If an entry is found then set the locator to the value of Locator in the control message. The next packet that is sent to that identifier will use the new locator.

b) If no mapping entry is found for the identifier then disregard the message or, optionally, create a new cache entry for the redirected identifier (this permits a mechanism for ILA routers to push mappings).

### 3.4 Other properties

Host No Mapping and Host Redirect are considered optional messages for hosts to send. If they are sent, then rate limiting should be a applied to avoid sending more than 100 control messages per second. If control messages are not generated, then a sender that has an out of date mapping should eventually timeout and stop sending packets to an incorrect locator. A host should not forward ILA addressed packets even in the case that it has the forwarding address for an identifier.

Router No Mapping and Router Redirect must be sent in order to keep hosts synchronized with mapping state. The number of messages sent to particular host should be rate limited so that no more than 100 messages per second are sent. An ILA router may proactively send Router No Mapping and Router Redirect messages upon a mapping state change to those hosts that are known to possess a mapping (the list of hosts that have sent packets through the router using a mapping could be kept in the mapping database). Conceivably, an ILA router could also multicast messages to inform hosts of a state (the details for this are outside the scope of this document).

## 4 Security Considerations

ILA control messages convey sensitive control information so they need to be protected against spoofing. ILA control messages should be authenticated and may be encrypted.

### 4.1 HMAC authentication and integrity

ILA control messages may optionally use HMAC authentication. The use of HMAC in a message is indicated by the H bit being set.

#### 4.1.1 Security fields in ILA control messages

This section summarizes the use of specific fields in the ILA messages. They are based on a key-hashed message authentication code (HMAC).

The security-related fields in ILA messages are:

- HMAC Key-id, 8 bits wide;
- HMAC, 256 bits wide (optional, exists only if HMAC Key-id is not 0).

The HMAC field is the output of the HMAC computation (per RFC 2104 [RFC2104]) using a pre-shared key and hashing algorithm identified by HMAC Key-id and of the text which consists of the concatenation of:

- the source IPv6 address;
- the ILA control message header and data

The purpose of the HMAC field is to verify the validity, the integrity and the authorization of the ILA control messages itself. If an outsider of the ILA domain does not have access to a current pre-shared secret, then it cannot compute the right HMAC so when the receiver of a control messages checks the validity of the HMAC it will simply reject the packet.

The HMAC Key-id field allows for the simultaneous existence of several hash algorithms (SHA-256, SHA3-256 ... or future ones) as well as pre-shared keys. This allows for pre-shared key roll-over when two pre-shared keys are supported for a while when all ILA nodes converged to a fresher pre-shared key. The HMAC Key-id field is opaque, i.e., it has neither syntax not semantic except as an index to the right combination of pre-shared key and hash algorithm and except that a value of 0 means that there is no HMAC field. It could also allow for interoperation among different ILA domains if allowed by local policy and assuming a collision-free Key Id allocation which is out of scope of this memo.

#### 4.1.2 Selecting a hash algorithm

The HMAC field in the ILA control messages is 256 bits wide. Therefore, the HMAC MUST be based on a hash function whose output is at least 256 bits. If the output of the hash function is 256, then this output is simply inserted in the HMAC field. If the output of the hash function is larger than 256 bits, then the output value is truncated to 256 by taking the least-significant 256 bits and

inserting them in the HMAC field.

ILA implementations can support multiple hash functions but MUST implement SHA-2 [FIPS180-4] in its SHA-256 variant.

#### 4.1.3 Pre-shared key management

The field HMAC Key-id allows for:

- key roll-over: when there is a need to change the key (the hash pre-shared secret), then multiple pre-shared keys can be used simultaneously. The validating routing can have a table of <HMAC Key-id, pre-shared secret, hash algorithm> for the currently active and future keys.

- different algorithm: by extending the previous table to <HMAC Key-id, hash function, pre-shared secret>, the validating router can also support simultaneously several hash algorithms (see section Section 4.1.2)

The pre-shared secret distribution can be done:

- in the configuration of the validating nodes, either by static configuration or any SDN oriented approach;

- dynamically using a trusted key distribution such as [RFC6407]

  The intent of this document is NOT to define yet-another-key-distribution-protocol.

### 4.2 DTLS

For stronger security, including encryption of ILA control messages, DTLS may be used. The use of DTLS necessitates a separate UDP port.

## 5 IANA Considerations

IANA will be requested to assign one UDP port number for ILA control messages, and one for ILA control messages with DTLS.

## 6 References

### 6.1 Normative References

[I-D.herbert-nvo3-ila] Herbert, T., "Identifier-locator addressing for network virtualization", draft-herbert-nvo3-ila-02 (work in progress), March 2016.

### 6.2 Informative References

[I-D.lapukhov-ila-deployment] Lapukhov, P., "Deploying Identifier-Locator Addressing (ILA) in datacenter", draft-lapukhov-ila-deployment-00 (work in progress), March 2016.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[RFC6407] Weis, B., Rowles, S., and T. Hardjono, "The Group Domain of Interpretation", RFC 6407, DOI 10.17487/RFC6407, October 2011, <http://www.rfc-editor.org/info/rfc6407>.

## Authors' Addresses

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA
US

EMail: tom@herbertland.com

INTERNET-DRAFT
Intended Status: Informational
Expires: September 14, 2017

Tom Herbert
Quantonium
Petr Lapukhov
Facebook
March 13, 2017

# Identifier-locator addressing for IPv6
## draft-herbert-nvo3-ila-04

Abstract

This specification describes identifier-locator addressing (ILA) for IPv6. Identifier-locator addressing differentiates between location and identity of a network node. Part of an address expresses the immutable identity of the node, and another part indicates the location of the node which can be dynamic. Identifier-locator addressing can be used to efficiently implement overlay networks for network virtualization as well as solutions for use cases in mobility.

Table of Contents

# 1 Introduction

This specification describes the address formats, protocol operation, and communication scenarios of identifier-locator addressing (ILA). In identifier-locator addressing, an IPv6 address is split into a locator and an identifier component. The locator indicates the topological location in the network for a node, and the identifier indicates the node’s identity which refers to the logical or virtual node in communications. Locators are routable within a network, but identifiers typically are not. An application addresses a peer destination by identifier. Identifiers are mapped to locators for transit in the network. The on-the-wire address is composed of a locator and an identifier: the locator is sufficient to route the packet to a physical host, and the identifier allows the receiving host to translate and forward the packet to the addressed application.

With identifier-locator addressing network virtualization and addressing for mobility can be implemented in an IPv6 network without any additional encapsulation headers. Packets sent with identifier-locator addresses look like plain unencapsulated packets (e.g. TCP/IP packets). This method is transparent to the network, so protocol specific mechanisms in network hardware work seamlessly. These mechanisms include hash calculation for ECMP, NIC large segment offload, checksum offload, etc.

Many of the concepts for ILA are adapted from Identifier-Locator Network Protocol (ILNP) ([RFC6740], [RFC6741]) which defines a protocol and operations model for identifier-locator addressing in IPv6.

Section 5 provides a motivation for ILA and comparison of ILA with alternative methods that achieve similar functionality.

## 1.1 Terminology

| | |
|---|---|
| ILA | Identifier-locator addressing. |
| ILA router | A network node that performs ILA translation and forwarding of translated packets. |
| ILA host | An end host that is capable of performing ILA translations on transmit or receive. |
| ILA node | A network node capable of performing ILA translations. This can be an ILA router or ILA host. |
| Locator | A network prefix that routes to a physical host. |

Locators provide the topological location of an addressed node. In ILA locators are a sixty-four bit prefixes.

Identifier A number that identifies an addressable node in the network independent of its location. ILA identifiers are sixty-four bit values.

ILA address
An IPv6 address composed of a locator (upper sixty-four bits) and an identifier (low order sixty-four bits).

SIR Standard identifier representation.

SIR prefix A sixty-four bit network prefix used to identify a SIR address.

SIR address
An IPv6 address composed of a SIR prefix (upper sixty-four bits) and an identifier (lower sixty-four bits). SIR addresses are visible to applications and provide a means to address nodes independent of their location.

SIR domain A unique identifier namespace defined by a SIR prefix. Each SIR prefix defines a SIR domain.

ILA translation
The process of translating the upper sixty-four bits of an IPv6 address. Translations may be from a SIR prefix to a locator or a locator to a SIR prefix.

Virtual address
An IPv6 or IPv4 address that resides in the address space of a virtual network. Such addresses may be translated to SIR addresses as an external representation of the address outside of the virtual network, or they may be translated to ILA addresses for transit over an underlay network.

Topological address
An address that refers to a non-virtual node in a network topology. These address physical hosts in a network.

## 2 Architectural overview

Identifier-locator addressing allows a data plane method to implement network virtualization without encapsulation and its related overheads. The service ILA provides is effectively layer 3 over layer 3 network virtualization (IPv4 or IPv6 over IPv6).

### 2.1 Addressing

ILA performs translations on IPv6 address. There are two types of addresses introduced for ILA: ILA addresses and SIR addresses.

ILA addresses are IPv6 addresses that are composed of a locator (upper sixty-four bits) and an identifier (low order sixty-four bits). The identifier serves as the logical addresses of a node, and the locator indicates the location of the node on the network.

A SIR address (standard identifier representation) is an IPv6 address that contains an identifier and an application visible SIR prefix. SIR addresses are visible to the application and can be used as connection endpoints. When a packet is sent to a SIR address, an ILA router or host overwrites the SIR prefix with a locator corresponding to the identifier. When a peer ILA node receives the packet, the locator is overwritten with the original SIR prefix before delivery to the application. In this manner applications only see SIR addresses, they do not have visibility into ILA addresses.

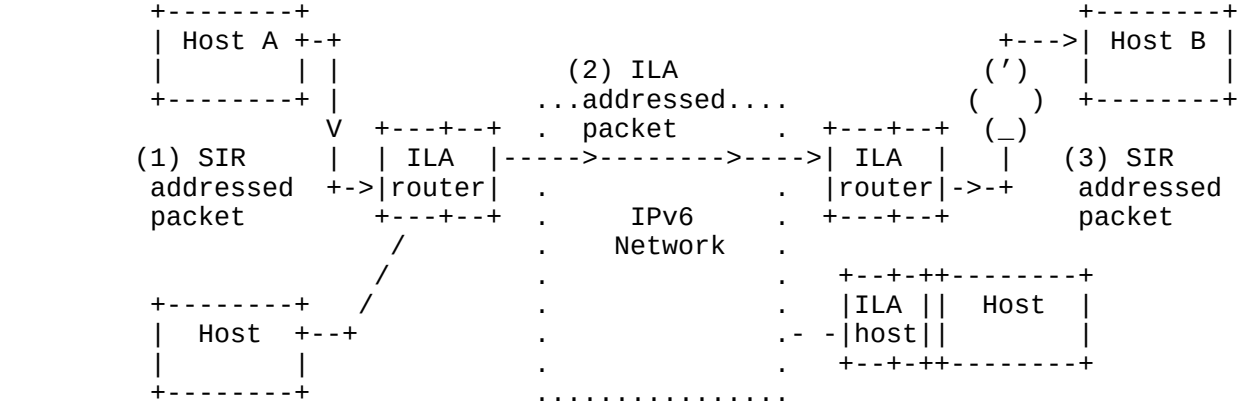
ILA translations can transform addresses from one type to another. In network virtualization virtual addresses can be translated into ILA and SIR addresses, and conversely ILA and SIR addresses can be translated to virtual addresses.

### 2.2 Network topology

ILA nodes are nodes in the network that perform ILA translations. An ILA router is a node that performs ILA address translation and packet forwarding to implement overlay network functionality. ILA routers perform translations on packets sent by end nodes for transport across an underlay network. Packets received by ILA routers on the underlay network have their addresses reversed translated for reception at an end node. An ILA host is an end node that implements ILA functionality for transmitting or receiving packets.

ILA nodes are responsible for transit of packets over an underlay network. On ingress to an ILA node (host or router) the virtual or SIR address of a destination is translated to an ILA address. At the a peer ILA node, the reverse translation is performed before handing packets to an application.

The figure below provides an example topology using ILA. ILA translations performed in one direction between Host A and Host B are denoted. Host A sends a packet with a destination SIR address (step (1)). An ILA router in the path translates the SIR address to an ILA address with a locator set to Host B, referring to the location of the node indicated by the identifier in the SIR address. The packet is forwarded over the network and delivered to a peer ILA node (step 2). The peer ILA node, in this case another ILA router, translates the destination address back to a SIR address and forwards to the final destination (step 3).




## 2.3 Translations and mappings

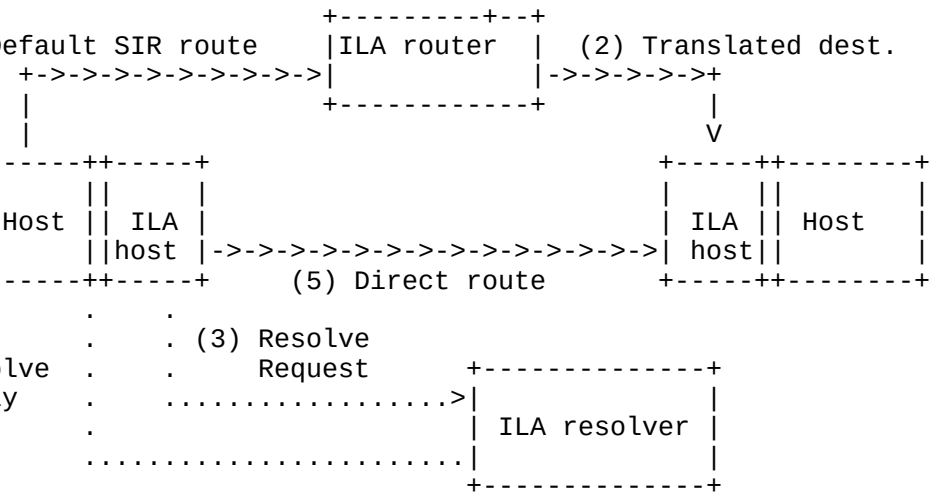
Address translation is the mechanism employed by ILA. Logical or virtual addresses are translated to topological IPv6 addresses for transport to the proper destination. Translation occurs in the upper sixty-four bits of an address, the low order sixty-four bits contains an identifier that is immutable and is not used to route a packet.

Each ILA node maintains a mapping table. This table maps identifiers to locators. The mappings are dynamic as nodes with identifiers can be created, destroyed, or migrated between physical hosts. Mappings are propagated amongst ILA routers or hosts in a network using mapping propagation protocols (mapping propagation protocols will be described in other specifications).

Identifiers are not statically bound to a host on the network, and in fact their binding (or location) may change. This is the basis for network virtualization and address migration. An identifier is mapped to a locator at any given time, and a set of identifier to locator mappings is propagated throughout a network to allow communications. The mappings are kept synchronized so that if an identifier migrates to a new physical host, its identifier to locator mapping is updated.

## 2.4 ILA routing

ILA is intended to be sufficiently lightweight so that all the hosts in a network could potentially send and receive ILA addressed packets. In order to scale this model and allow for hosts that do not participate in ILA, a routing topology may be applied. A simple routing topology is illustrated below.




An ILA router can be addressed by an "anycast" SIR prefix so that it receives packets sent on the network with SIR addresses. When an ILA router receives a SIR addressed packet (step (1) in the diagram) it will perform the ILA translation and send the ILA addressed packet to the destination ILA node (step (2)).

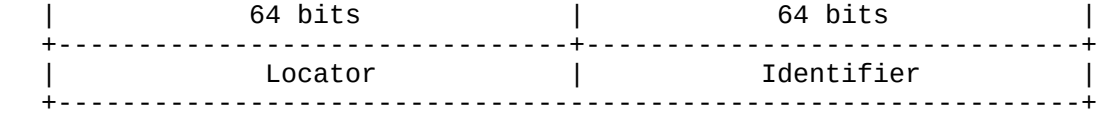
If a sending host is ILA capable the triangular routing can be eliminated by performing an ILA resolution protocol. This entails the host sending an ILA resolve request that specifies the SIR address to resolve (step (3) in the figure). An ILA resolver can respond to a resolver request with the identifier to locator mapping (step (4)). Subsequently, the ILA host can perform ILA translation and send directly to the destination specified in the locator (step (5) in the figure). The ILA resolution protocol will be specified in a companion document.

In this model an ILA host maintains a cache of identifier mappings for identifiers that it is currently communicating with. ILA routers are expected to maintain a complete list of identifier to locator mappings within the SIR domains that they service.

# 3 Address formats

## 3.1 ILA address format

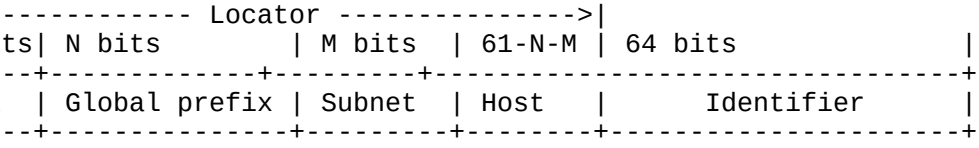
An ILA address is composed of a locator and an identifier where each occupies sixty-four bits (similar to the encoding in ILNP [RFC6741]).



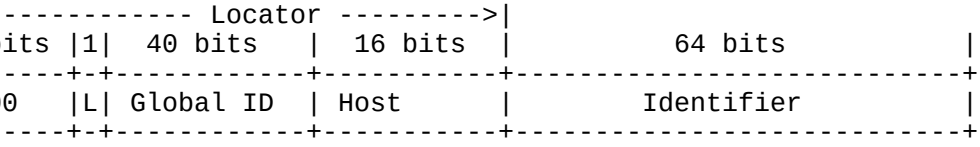

## 3.2 Locators

Locators are routable network address prefixes that create topological addresses for physical hosts within the network. They may be assigned from a global address block [RFC3587], or be based on unique local IPv6 unicast addresses as described in [RFC4193].

The format of an ILA address with a global unicast locator is:

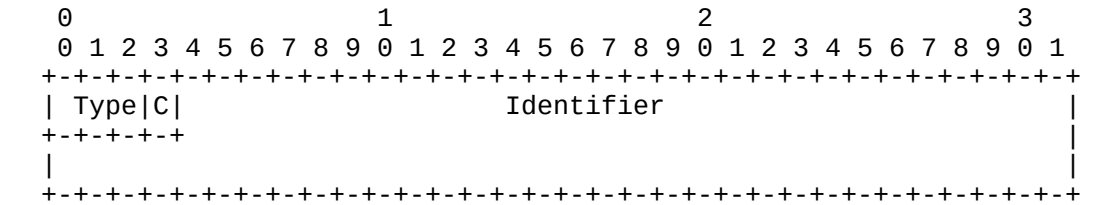



The format of an ILA address with a unique local IPv6 unicast locator is:




## 3.3 Identifiers

The format of an ILA identifier is:

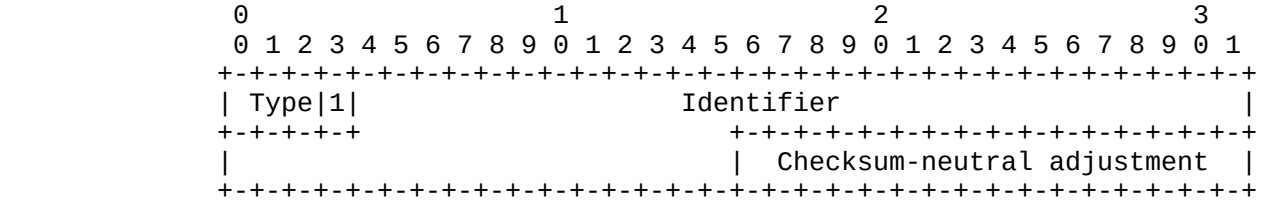


Fields are:

- Type: Type of the identifier (see section 3.3.2).
- C: The C-bit. This indicates that checksum-neutral mapping applied (see section 3.3.1).
- Identifier: Identifier value.

### 3.3.1 Checksum neutral-mapping format

If the C-bit is set the low order sixteen bits of an identifier contain the adjustment for checksum-neutral mapping (see section 4.4.1 for description of checksum-neutral mapping). The format of an identifier with checksum neutral mapping is:




### 3.3.2 Identifier types

Identifier types allow standard encodings for common uses of identifiers. Defined identifier types are:

0: interface identifier

1: locally unique identifier

2: virtual networking identifier for IPv4 address

3: virtual networking identifier for IPv6 unicast address

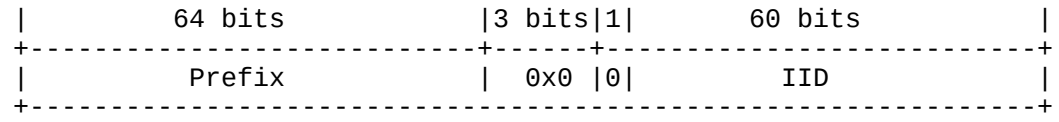
4: virtual networking identifier for IPv6 multicast address

5-7: Reserved

#### 3.3.2.1 Interface identifiers

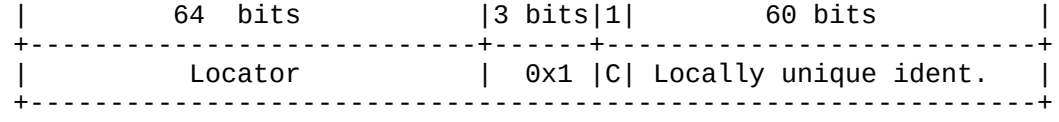
The interface identifier type indicates a plain local scope interface identifier. When this type is used the address is a normal IPv6 address without identifier-locator semantics. The purpose of this type is to allow normal IPv6 addresses to be defined within the same networking prefix as ILA addresses. Type bits and C-bit MUST be zero.

The format of an ILA interface identifier address is:

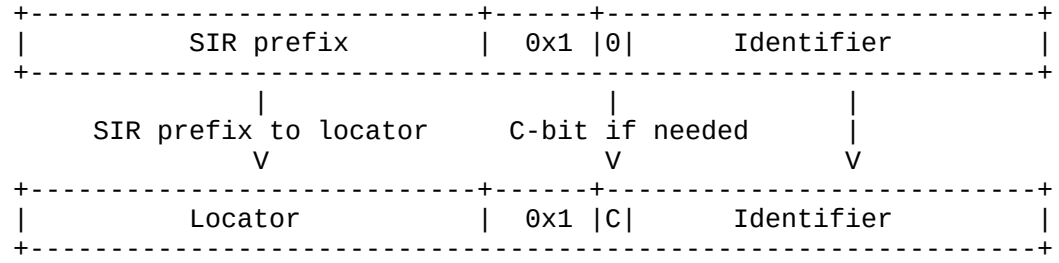



3.3.2.2 Locally unique identifiers

Locally unique identifiers (LUI) can be created for various addressable objects within a network. These identifiers are in a flat sixty bit space and must be unique within a SIR domain (unique within a site for instance). To simplify administration, hierarchical allocation of locally unique identifiers may be performed. The format of an ILA address with locally unique identifiers is:

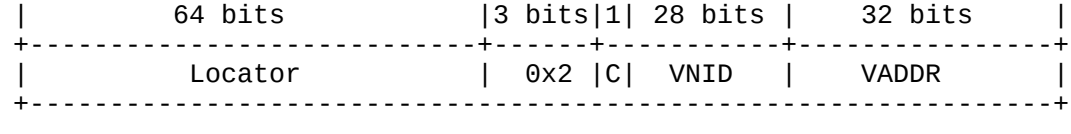



The figure below illustrates the translation from SIR address to an ILA address as would be performed when a node sends to a SIR address. Note the low order 16 bites of the identifier may be modified as the checksum-neutral adjustment. The reverse translation of ILA address to SIR address is symmetric.



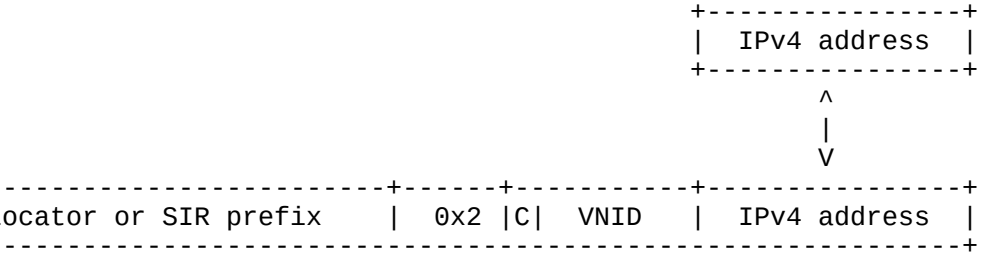

3.3.2.3 Virtual networking identifiers for IPv4

This type defines a format for encoding an IPv4 virtual address and virtual network identifier within an identifier. The format of an ILA address for IPv4 virtual networking is:



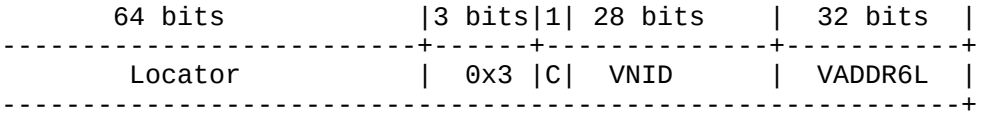
VNID is a virtual network identifier and VADDR is a virtual address within the virtual network indicated by the VNID. The VADDR can be an IPv4 unicast or multicast address, and may often be in a private address space (i.e. [RFC1918]) used in the virtual network.

Translating a virtual IPv4 address into an ILA or SIR address and the reverse translation are straight forward. Note that the low order 16 bits of the IPv6 address may be modified as the checksum-neutral adjustment and that this translation implies protocol translation when sending IPv4 packets over an ILA IPv6 network.



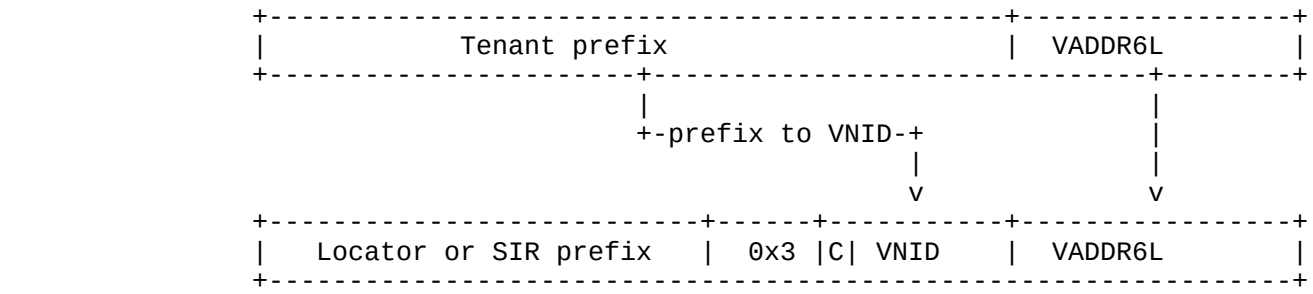

3.3.2.4 Virtual networking identifiers for IPv6 unicast

In this format, a virtual network identifier and virtual IPv6 unicast address are encoded within an identifier. To facilitate encoding of virtual addresses, there is a unique mapping between a VNID and a ninety-six bit prefix of the virtual address. The format an IPv6 unicast encoding with VNID in an ILA address is:

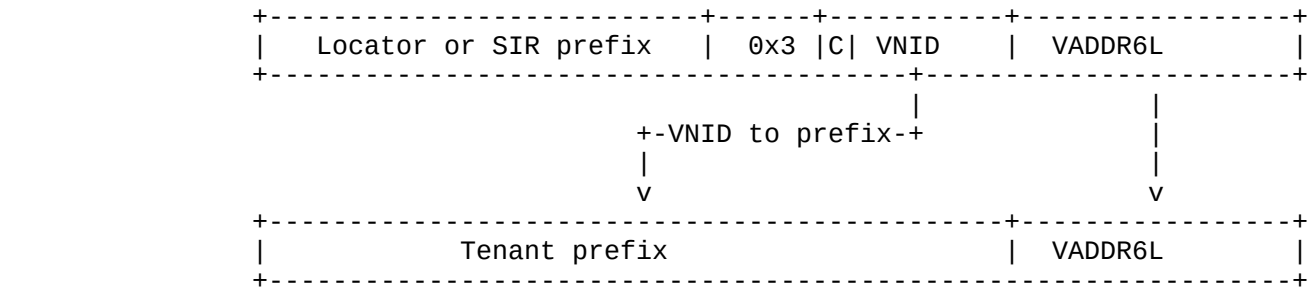



VADDR6L contains the low order 32 bits of the IPv6 virtual address. The upper 96 bits of the virtual address are inferred from the VNID to prefix mapping. Note that for ILA translations the low order sixteen of the VADDR6L may be modified for checksum-neutral adjustment.

The figure below illustrates encoding a tenant IPv6 virtual unicast address into a ILA or SIR address.

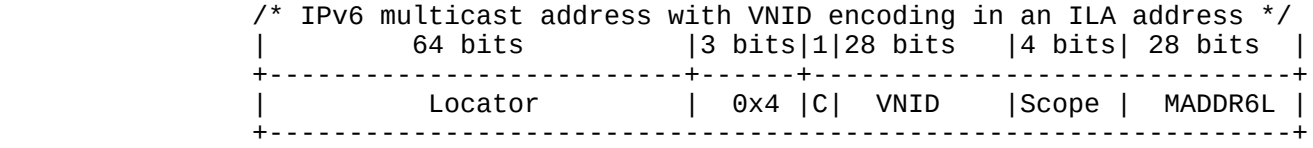



This encoding is reversible, given an ILA address, the virtual address visible to the tenant can be deduced:




3.3.2.5 Virtual networking identifiers for IPv6 multicast

In this format, a virtual network identifier and virtual IPv6 multicast address are encoded within an identifier.




This format encodes an IPv6 multicast address in an identifier. The scope indicates multicast address scope as defined in [RFC7346]. MADDR6L is the low order 28 bits of the multicast address. The full multicast address is thus:

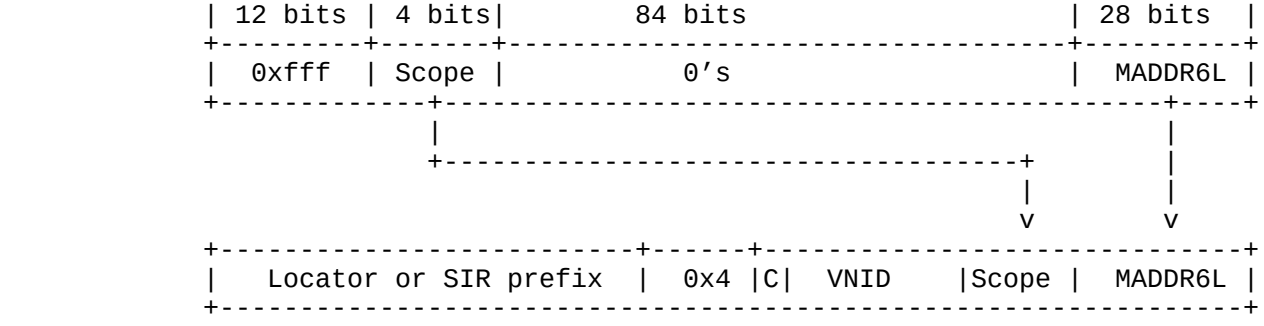
ff0<Scope>::<MADDRL6 high 12 bits>:<MADDRL6 low 16 bits>

And so can encode multicast addresses of the form:

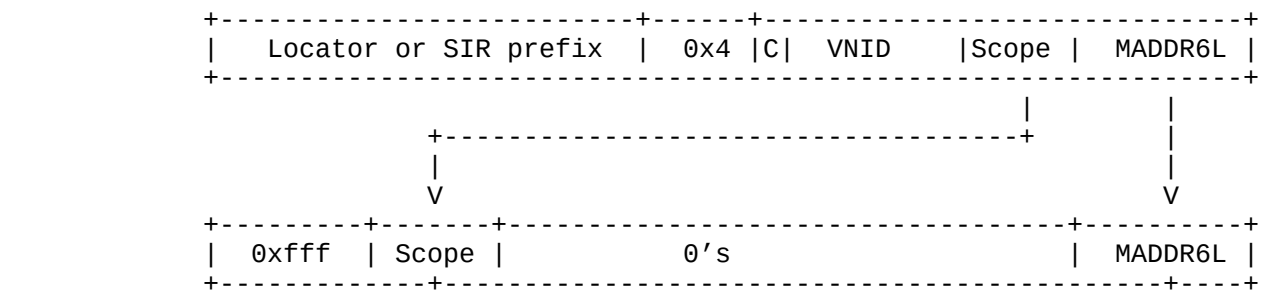
ff0X::0 to ff0X::0fff:ffff

The figure below illustrates encoding a tenant IPv6 virtual multicast

address in an ILA or SIR address. Note that low order sixteen bits of MADDR6L may be modified to be the checksum-neutral adjustment.



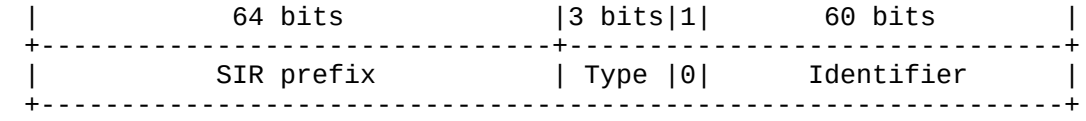

This translation is reversible:




## 3.4 Standard identifier representation addresses

An identifier identifies objects or nodes in a network. For instance, an identifier may refer to a specific host, virtual machine, or tenant system. When a host initiates a connection or sends a packet, it uses the identifier to indicate the peer endpoint of the communication. The endpoints of an established connection context also referenced by identifiers. It is only when the packet is actually being sent over a network that the locator for the identifier needs to be resolved.

In order to maintain compatibility with existing networking stacks and applications, identifiers are encoded in IPv6 addresses using a standard identifier representation (SIR) address. A SIR address is a combination of a prefix which occupies what would be the locator portion of an ILA address, and the identifier in its usual location. The format of a SIR address is:




The C-bit (checksum-neutral mapping) MUST be zero for a SIR address. Type may be any identifier type except zero (interface identifiers)

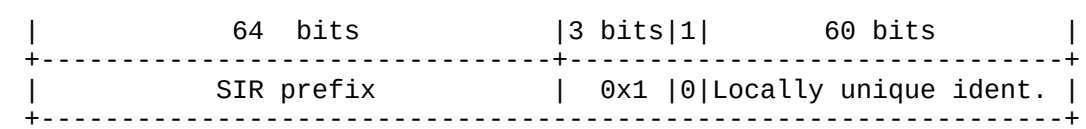
A SIR prefix may be site-local, or globally routable. A globally routable SIR prefix facilitates connectivity between hosts on the Internet and ILA nodes. A gateway between a site's network and the Internet can translate between SIR prefix and locator for an identifier. A network may have multiple SIR prefixes where each prefix defines a unique identifier space.

Locators MUST only be associated with one SIR prefix. This ensures that if a translation from a SIR address to an ILA address is performed when sending a packet, the reverse translation at the receiver yields the same SIR address that was seen at the transmitter. This also ensures that a reverse checksum-neutral mapping can be performed at a receiver to restore the addresses that were included in a pseudo header for setting a transport checksum.

A standard identifier representation address can be used as the externally visible address for a node. This can used throughout the network, returned in DNS AAAA records [RFC3363], used in logging, etc. An application can use a SIR address without knowledge that it encodes an identifier.

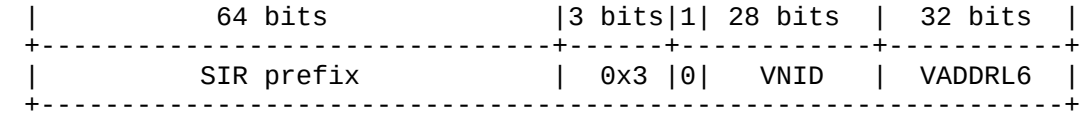
### 3.4.1 SIR for locally unique identifiers

The SIR address for a locally unique identifier has format:




### 3.4.2 SIR for virtual addresses

A virtual address can be encoded using the standard identifier representation. For example, the SIR address for an IPv6 virtual address may be:



Note that this allows three representations of the same address in he network: as a virtual address, a SIR address, and an ILA address.

### 3.4.3 SIR domains

Each SIR prefix defines a SIR domain. A SIR domain is a unique name space for identifiers within a domain. The full identity of a node is thus determined by an identifier and SIR domain (SIR prefix). Locators MUST map to only one SIR domain in order to ensure that translation from a locator to SIR prefix is unambiguous.

## 4 Operation

This section describes operation methods for using identifier-locator addressing.

### 4.1 Identifier to locator mapping

An application initiates a communication or flow using a SIR address or virtual address for a destination. In order to send a packet on the network, the destination address is translated by an ILA router or an ILA host in the path. An ILA node maintains a list of mappings from identifier to locator to perform this translation.

The mechanisms of propagating and maintaining identifier to locator mappings are outside the scope of this document.

### 4.2 Address translations

With ILA, address translation is performed to convert SIR addresses to ILA addresses, and ILA addresses to SIR addresses. Translation is usually done on a destination address as a form of source routing, however translation on source virtual addresses to SIR addresses can also be done to support some network virtualization scenarios (see appendix A.7 for example).

#### 4.2.1 SIR to ILA address translation

When translating a SIR address to an ILA address the SIR prefix in the address is overridden with a locator, and checksum neutral mapping may be performed. Since this operation is potentially done for every packet the process should be very efficient (particularly the lookup and checksum processing operations).

The typical steps to transmit a packet using ILA are:

1) Host stack creates a packet with source address set to a local address (possibly a SIR address) for the local identity, and

the destination address is set to the SIR address or virtual address for the peer. The peer address may have been discovered through DNS or other means.

2) An ILA router or host translates the packet to use the locator. If the original destination address is a SIR address then the SIR prefix is overwritten with the locator. If the original packet is a virtually addressed tenant packet then the virtual address is translated per section 3.3.2. The locator is discovered by a lookup in the locator to identifier mappings.

3) The ILA node performs checksum-neutral mapping if configured for that (section 4.4.1).

4) Packet is forwarded on the wire. The network routes the packet to the host indicated by the locator.

#### 4.2.2 ILA to SIR address translation

When a destination node (ILA router or end host) receives an ILA addressed packet, the ILA address MUST be translated back to a SIR address (or tenant address) before upper layer processing.

The steps of receive processing are:

1) Packet is received. The destination locator is verified to match a locator assigned to the host.

2) A lookup is performed on the destination identifier to find if it addresses a local identifier. If match is found, either the locator is overwritten with SIR prefix (for locally unique identifier type) or the address is translated back to a tenant virtual address as shown in appendix A.7.

3) Perform reverse checksum-neutral mapping if C-bit is set (section 4.4.1).

4) Perform any optional policy checks; for instance that the source may send a packet to the destination address, that packet is not illegitimately crossing virtual networks, etc.

5) Forward packet to application processing.

### 4.3 Virtual networking operation

When using ILA with virtual networking identifiers, address translation is performed to convert tenant virtual network and virtual addresses to ILA addresses, and ILA addresses back to a

virtual network and tenant's virtual addresses. Translation may occur on either source address, destination address, or both (see scenarios for virtual networking in Appendix A). Address translation is performed similar to the SIR translation cases described above.

#### 4.3.1 Crossing virtual networks

With explicit configuration, virtual network hosts may communicate directly with virtual hosts in another virtual network by using SIR addresses for virtualization in both the source and destination addresses. This might be done to allow services in one virtual network to be accessed from another (by prior agreement between tenants). See appendix A.13 for example of ILA addressing for such a scenario.

#### 4.3.2 IPv4/IPv6 protocol translation

An IPv4 tenant may send a packet that is converted to an IPv6 packet with ILA addresses. Similarly, an IPv6 packet with ILA addresses may be converted to an IPv4 packet to be received by an IPv4-only tenant. These are IPv4/IPv6 stateless protocol translations as described in [RFC6144] and [RFC6145]. See appendix A.12 for a description of these scenarios.

### 4.4 Transport layer checksums

Packets undergoing ILA translation may encapsulate transport layer checksums (e.g. TCP or UDP) that include a pseudo header that is affected by the translation.

ILA provides two alternatives do deal with this:

- Perform a checksum-neutral mapping to ensure that an encapsulated transport layer checksum is kept correct on the wire.

- Send the checksum as-is, that is send the checksum value based on the pseudo header before translation.

Some intermediate devices that are not the actual end point of a transport protocol may attempt to validate transport layer checksums. In particular, many Network Interface Cards (NICs) have offload capabilities to validate transport layer checksums (including any pseudo header) and return a result of validation to the host. Typically, these devices will not drop packets with bad checksums, they just pass a result to the host. Checksum offload is a performance benefit, so if packets have incorrect checksums on the wire this benefit is lost. With this incentive, applying a checksum-

neutral mapping is the recommended alternative. If it is known that the addresses of a packet are not included in a transport checksum, for instance a GRE packet is being encapsulated, then a source may choose not to perform checksum-neutral mapping.

#### 4.4.1 Checksum-neutral mapping

When a change is made to one of the IP header fields in the IPv6 pseudo-header checksum (such as one of the IP addresses), the checksum field in the transport layer header may become invalid. Fortunately, an incremental change in the area covered by the Internet standard checksum [RFC1071] will result in a well-defined change to the checksum value [RFC1624]. So, a checksum change caused by modifying part of the area covered by the checksum can be corrected by making a complementary change to a different 16-bit field covered by the same checksum.

ILA can perform a checksum-neutral mapping when a SIR prefix or virtual address is translated to a locator in an IPv6 address, and performs the reverse mapping when translating a locator back to a SIR prefix or virtual address. The low order sixteen bits of the identifier contain the checksum adjustment value for ILA.

On transmission, the translation process is:

1) Compute the one’s complement difference between the SIR prefix and the locator. Fold this value to 16 bits (add-with-carry four 16-bit words of the difference).

2) Add-with-carry the bit-wise not of the 0x1000 (i.e. 0xefff) to the value from #1. This compensates the checksum for setting the C-bit.

3) Add-with-carry the value from #2 to the low order sixteen bits of the identifier.

4) Set the resultant value from #3 in the low order sixteen bits of the identifier and set the C-bit.

Note that the "adjustment" (the 16-bit value set in the identifier in set #3) is fixed for a given SIR to locator mapping, so the adjustment value can be saved in an associated data structure for a mapping to avoid computing it for each translation.

On reception of an ILA addressed packet, if the C-bit is set in an ILA address:

1) Compute the one’s complement difference between the locator in

the address and the SIR prefix that the locator is being translated to. Fold this value to 16 bits (add-with-carry four 16-bit words of the difference).

2) Add-with-carry 0x1000 to the value from #1. This compensates the checksum for clearing the C-bit.

3) Add-with-carry the value from #2 to the low order sixteen bits of the identifier.

4) Set the resultant value from #3 in the low order sixteen bits of the identifier and clear the C-bit. This restores the original identifier sent in the packet.

#### 4.4.2 Sending an unmodified checksum

When sending an unmodified checksum, the checksum is incorrect as viewed in the packet on the wire. At the receiver, ILA translation of the destination ILA address back to the SIR address occurs before transport layer processing. This ensures that the checksum can be verified when processing the transport layer header containing the checksum. Intermediate devices are not expected to drop packets due to a bad transport layer checksum.

### 4.5 Address selection

There may be multiple possibilities for creating either a source or destination address. A node may be associated with more than one identifier, and there may be multiple locators for a particular identifier. The choice of locator or identifier is implementation or configuration specific. The selection of an identifier occurs at flow creation and must be invariant for the duration of the flow. Locator selection must be done at least once per flow, and the locator associated with the destination of a flow may change during the lifetime of the flow (for instance in the case of a migrating connection it will change). ILA address selection should follow specifications in Default Address Selection for Internet Protocol Version 6 (IPv6) [RFC6724].

### 4.6 Duplicate identifier detection

As part of implementing the locator to identifier mapping, duplicate identifier detection should be implemented in a centralized control plane. A registry of identifiers could be maintained (possibly in association the identifier to locator mapping database). When a node creates an identifier it registers the identifier, and when the identifier is no longer in use (e.g. task completes) the identifier is unregistered. The control plane should able to detect a

registration attempt for an existing identifier and deny the request.

### 4.7 ICMP error handling

A packet that contains an ILA address may cause ICMP errors within the network. In this case the ICMP data contains an IP header with an ILA address. ICMP messages are sent back to the source address in the packet. Upon receiving an ICMP error the host will process it differently depending on whether it is ILA capable.

#### 4.7.1 Handling ICMP errors by ILA capable hosts

If a host is ILA capable it can attempt to reverse translate the ILA address in the destination of a header in the ICMP data back to a SIR address that was originally used to transmit the packet. The steps are:

1) Assume that the upper sixty-four bits of the destination address in the ICMP data is a locator. Try match these bits back to a SIR address. If the host is only in one SIR domain, then the mapping to SIR address is implicit. If the host is in multiple domains then a locator to SIR addresses table can be maintained for this lookup.

2) If the identifier is marked with checksum-neutral mapping, undo the checksum-neutral using the SIR address found in #1. The resulting identifier address is potentially the original address used to send the packet.

3) Lookup the identifier in the identifier to locator mapping table. If an entry is found compare the locator in the entry to the locator (upper sixty-four bits) of the destination address in the IP header of the ICMP data. If these match then proceed to next step.

4) Overwrite the upper sixty-four bits of the destination address in the ICMP data with the found SIR address and overwrite the low order sixty-four bits with the found identifier (the result of undoing checksum-neutral mapping). The resulting address should be the original SIR address used in sending. The ICMP error packet can then be received by the stack for further processing.

#### 4.7.2 Handling ICMP errors by non-ILA capable hosts

A non-ILA capable host may receive an ICMP error generated by the network that contains an ILA address in an IP header contained in the ICMP data. This would happen in the case that an ILA router performed

translation on a packet the host sent and that packet subsequently generated an ICMP error. In this case the host receiving the error message will attempt to find the connection state corresponding to the packet in headers the ICMP data. Since the host is unaware of ILA the lookup for connection state should fail. Because the host cannot recover the original addresses it used to send the packet, it won't be able any to derive any useful information about the original destination of the packet that it sent.

If packets for a flow are always routed through an ILA router in both directions, for example ILA routers are coincident with edge routes in a network, then ICMP errors could be intercepted by an intermediate node which could translate the destination addresses in ICMP data back to the original SIR addresses. A receiving host would then see the destination address in the packet of the ICMP data to be that it used to transmit the original packet.

### 4.8 Multicast

ILA is generally not intended for use with multicast. In the case of multicast, routing of packets is based on the source address. Neither the SIR address nor an ILA address is suitable for use as a source address in a multicast packet. A SIR address is unroutable and hence would make a multicast packet unroutable if used as a source address. Using an ILA address as the source address makes the multicast packet routable, but this exposes ILA address to applications which is especially problematic on a multicast receiver that doesn't support ILA.

If all multicast receivers are known to support ILA, a local locator address may be used in the source address of the multicast packet. In this case, each receiver will translate the source address from an ILA address to a SIR address before delivering packets to an application.

## 5 Motivation for ILA

### 5.1 Use cases

#### 5.1.1 Multi-tenant virtualization

In multi-tenant virtualization overlay networks are established for tenants to provide virtual networks. Each tenant may have one or more virtual networks and a tenant's nodes are assigned virtual addresses within virtual networks. Identifier-locator addressing may be used as an alternative to traditional network virtualization encapsulation protocols used to create overlay networks (e.g. VXLAN [RFC7348]). Section 5.2.4 describes the advantages of using ILA in lieu of

encapsulation protocols.

Tenant systems (e.g. VMs) run on physical hosts and may migrate to different hosts. A tenant system is identified by a virtual address and virtual networking identifier of a corresponding virtual network. ILA can encode the virtual address and a virtual networking identifier in an ILA identifier. Each identifier is mapped to a locator that indicates the current host where the tenant system resides. Nodes that send to the tenant system set the locator per the mapping. When a tenant system migrates its identifier to locator mapping is updated and communicating nodes will use the new mapping.

#### 5.1.2 Datacenter virtualization

Datacenter virtualization virtualizes networking resources. Various objects within a datacenter can be assigned addresses and serve as logical endpoints of communication. A large address space, for example that of IPv6, allows addressing to be used beyond the traditional concepts of host based addressing. Addressed objects can include tasks, virtual IP addresses (VIPs), pieces of content, disk blocks, etc. Each object has a location which is given by the host on which an object resides. Some objects may be migratable between hosts such that their location changes over time.

Objects are identified by a unique identifier within a namespace for the datacenter (appendix B discusses methods to create unique identifiers for ILA). Each identifier is mapped to a locator that indicates the current host where the object resides. Nodes that send to an object set the locator per the mapping. When an object migrates its identifier to locator mapping is updated and communicating nodes will use the new mapping.

A datacenter object of particular interest is tasks, units of execution for for applications. The goal of virtualzing tasks is to maximize resource efficiency and job scheduling. Tasks share many properties of tenant systems, however they are finer grained objects, may have a shorter lifetimes, and are likely created in greater numbers. Appendix C provides more detail and motivation for virtualizing tasks using ILA.

#### 5.1.3 Device mobility

ILA may be applied as a solution for mobile devices. These are devices, smart phones for instance, that physically move between different networks. The goal of mobility is to provide a seamless transition when a device moves from one network to another.

Each mobile device is identified by unique identifier within some

provider domain. ILA encodes the identifier for the device in an ILA identifier. Each identifier is mapped to a locator that indicates the current network or point of attachment for the device. Nodes that send to the device set the locator per the mapping. When a mobile device moves between networks its identifier to locator mapping is updated and communicating nodes will use the new mapping.

### 5.2 Alternative methods

This section discusses the merits of alternative solution that have been proposed to provide network virtualization or mobility in IPv6.

#### 5.2.1 ILNP

ILNP splits an address into a locator and identifier in the same manner as ILA. ILNP has characteristics, not present in ILA, that prevent it from being a practical solution:

- o ILNP requires that transport layer protocol implementations must be modified to work over ILNP.
- o ILNP can only be implemented in end hosts, not within the network. This essentially requires that all end hosts need to be modified to participate in mobility.
- o ILNP employs IPv6 extension headers which are mostly considered non-deployable. ILA does not use these.
- o Core support for ILA is in upstream Linux, to date there is no publicly available source code for ILNP.
- o ILNP involves DNS to distribute mapping information, ILA assumes mapping information is not part of naming.

#### 5.2.2 Flow label as virtual network identifier

The IPv6 flow label could conceptually be used as a 20-bit virtual network identifier in order to indicate a packet is sent on an overlay network. In this model the addresses may be virtual addresses within the specified virtual network. Presumably, the tuple of flow-label and addresses could be used by switches to forward virtually addressed packets.

This approach has some issues:

- o Forwarding virtual packets to their physical location would require specialized switch support.

- The flow label is only twenty bits, this is too small to be a discriminator in forwarding a virtual packet to a specific destination. Conceptually, the flow label might be used in a type of label switching to solve that.

- The flow label is not considered immutable in transit, intermediate devices may change it.

- The flow label is not part of the pseudo header for transport checksum calculation, so it is not covered by any transport (or other) checksums.

#### 5.2.3 Extension headers

To accomplish network virtualization an extension header, as a destination or routing option, could be used that contains the virtual destination address of a packet. The destination address in the IPv6 header would be the topological address for the location of the virtual node. Conceivably, segment routing could be used to implement network virtualization in this manner.

This technique has some issues:

- Intermediate devices must not insert extension headers [RFC2460bis].

- Extension headers introduce additional packet overhead which may impact performance.

- Extension headers are not covered by transport checksums (as the addresses would be) nor any other checksum.

- Extension headers are not widely supported in network hardware or devices. For instance, several NIC offloads don’t work in the presence of extension headers.

#### 5.2.4 Encapsulation techniques

Various encapsulation techniques have been proposed for implementing network virtualization and mobility. LISP is an example of an encapsulation that is based on locator identifier separation similar to ILA. The primary drawback of encapsulation is complexity and per packet overhead. For, instance when LISP is used with IPv6 the encapsulation overhead is fifty-six bytes and two IP headers are present in every packet. This adds considerable processing costs, requires considerations to handle path MTU correctly, and certain network accelerations may be lost.

## 6 IANA Considerations

There are no IANA considerations in this specification.

# 7 References

## 7.1 Normative References

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2460bis] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", draft-ietf-6man-rfc2460bis-03, January 2016.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, June 2011.

[RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.

[RFC1624] Rijsinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, September 2012.

## 7.2 Informative References

[RFC6740] RJ Atkinson and SN Bhatti, "Identifier-Locator Network Protocol (ILNP) Architectural Description", RFC 6740, November 2012.

[RFC6741] RJ Atkinson and SN Bhatti, "Identifier-Locator Network Protocol (ILNP) Engineering Considerations", RFC 6741, November 2012.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC3363] Bush, R., Durand, A., Fink, B., Gudmundsson, O., and T. Hain, "Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System (DNS)", RFC 3363, August 2002.

[RFC3587] Hinden, R., Deering, S., and E. Nordmark, "IPv6 Global

Unicast Address Format", RFC 3587, August 2003.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.

[RFC6144] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", RFC 6144, April 2011.

[NVO3ARCH] Black, D., Hudson, J., Kreeger, L., Lasserre, M., and Narten, T., "An Architecture for Overlay Networks (NVO3)", draft-ietf-nvo3-arch-03

[GUE] Herbert, T., and Yong, L., "Generic UDP Encapsulation", draft-herbert-gue-02, work in progress.

[GUESEC] Yong, L., and Herbert, T. "Generic UDP Encapsulation (GUE) for Secure Transport", draft-hy-gue-4-secure-transport-00, work in progress

## 8 Acknowledgments

The author would like to thank Mark Smith, Lucy Yong, Erik Kline, Saleem Bhatti, Petr Lapukhov, Blake Matheny, Doug Porter, Pierre Pfister, and Fred Baker for their insightful comments for this draft; Roy Bryant, Lorenzo Colitti, Mahesh Bandewar, and Erik Kline for their work on defining and applying ILA.

## Appendix A: Communication scenarios

This section describes the use of identifier-locator addressing in several scenarios.

### A.1 Terminology for scenario descriptions

A formal notation for identifier-locator addressing with ILNP is described in [RFC6740]. We extend this to include for network virtualization cases.

Basic terms are:

A = IP Address
I = Identifier
L = Locator
LUI = Locally unique identifier
VNI = Virtual network identifier
VA = An IPv4 or IPv6 virtual address
VAX = An IPv6 networking identifier (IPv6 VA mapped to VAX)
SIR = Prefix for standard identifier representation
VNET = IPv6 prefix for a tenant (assumed to be globally routable)
Iaddr = IPv6 address of an Internet host

An ILA IPv6 address is denoted by

L:I

A SIR address with a locally unique identifier and SIR prefix is denoted by

SIR:LUI

A virtual identifier with a virtual network identifier and a virtual IPv4 address is denoted by

VNI:VA

An ILA IPv6 address with a virtual networking identifier for IPv4 would then be denoted

L:(VNI:VA)

The local and remote address pair in a packet or endpoint is denoted

A,A

An address translation sequence from SIR addresses to ILA addresses

for transmission on the network and back to SIR addresses at a receiver has notation:

A,A -> L:I,A -> A,A

## A.2 Identifier objects

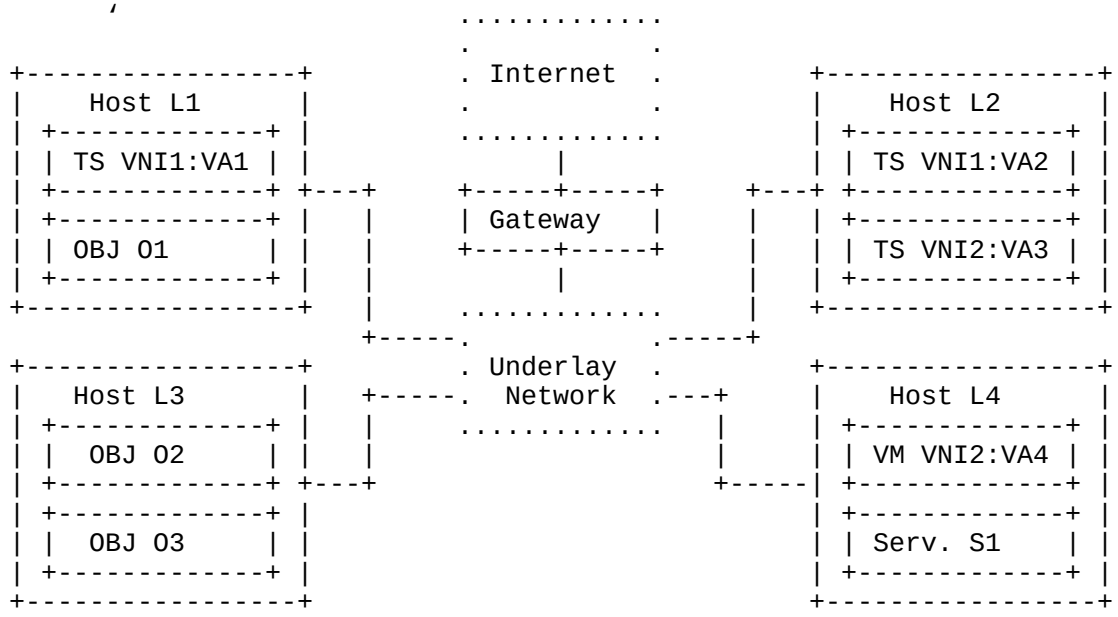
Identifier-locator addressing is broad enough in scope to address many different types of networking entities. For the purposes of this section we classify these as "objects" and "tenant systems".

Objects encompass uses where nodes are address by local unique identifiers (LUI). In the scenarios below objects are denoted by OBJ.

Tenant systems are those associated with network virtualization that have virtual addresses (that is they are addressed by VNI:VA). In the scenarios below tenant systems are denoted by TS.

## A.3 Reference network for scenarios

The figure below provides an example network topology with ILA addressing in use. In this example, there are four hosts in the network with locators L1, L2, L3, and L4. There three objects with identifiers O1, O2, and O3, as well as a common networking service with identifier S1. There are two virtual networks VNI1 and VNI2, and four tenant systems addressed as: VA1 and VA2 in VNI1, VA3 and VA4 in VNI2. The network is connected to the Internet via a gateway.



Several communication scenarios can be considered:

1) Object to object
2) Object to Internet
3) Internet to object
4) Tenant system to local service
5) Object to tenant system
6) Tenant system to Internet
7) Internet to tenant system
8) IPv4 tenant system to service
9) Tenant system to tenant system same virtual network using IPv6
10) Tenant system to tenant system in same virtual network using IPv4
11) Tenant system to tenant system in different virtual network using IPv6
12) Tenant system to tenant system in different virtual network using IPv4
13) IPv4 tenant system to IPv6 tenant system in different virtual networks

### A.4 Scenario 1: Object to task

The transport endpoints for object to object communication are the SIR addresses for the objects. When a packet is sent on the wire, the locator is set in the destination address of the packet. On reception the destination addresses is converted back to SIR representation for processing at the transport layer.

If object O1 is communicating with object O2, the ILA translation sequence would be:

```
SIR:O1,SIR:O2 ->                    // Transport endpoints on O1
SIR:O1,L3:O2 ->                     // ILA used on the wire
SIR:O1,SIR:O2                       // Received at O2
```

### A.5 Scenario 2: Object to Internet

Communication from an object to the Internet is accomplished through use of a SIR address (globally routable) in the source address of packets. No ILA translation is needed in this path.

If object O1 is sending to an address Iaddr on the Internet, the packet addresses would be:

```
SIR:O1,Iaddr
```

### A.6 Scenario 3: Internet to object

An Internet host transmits a packet to a task using an externally routable SIR address. The SIR prefix routes the packet to a gateway for the datacenter. The gateway translates the destination to an ILA address.

If a host on the Internet with address Iaddr sends a packet to object O3, the ILA translation sequence would be:

```
  Iaddr,SIR:O3 ->                    // Transport endpoint at Iaddr
  Iaddr,L1:O3 ->                     // On the wire in datacenter
  Iaddr,SIR:O3                       // Received at O3
```

### A.7 Scenario 4: Tenant system to service

A tenant can communicate with a datacenter service using the SIR address of the service.

If TS VA1 is communicating with service S1, the ILA translation sequence would be:

```
  VNET:VA1,Saddr->                   // Transport endpoints in TS
  SIR:(VNET:VA1):Saddr->             // On the wire
  SIR:(VNET:VA1):Saddr               // Received at S1
```

Where VNET is the address prefix for the tenant and Saddr is the IPv6 address of the service.

The ILA translation sequence in the reverse path, service to tenant system, would be:

```
  Saddr,SIR:(VNET:VA1)               // Transport endpoints in S1
  Saddr,L1:(VNET:VA1)                // On the wire
  Saddr,VNET:VA1                     // Received at the TS
```

Note that from the point of view of the service task there is no material difference between a peer that is a tenant system versus one which is another task.

### A.8 Scenario 5: Object to tenant system

An object can communicate with a tenant system through it’s externally visible address.

If object O2 is communicating with TS VA4, the ILA translation sequence would be:

```
  SIR:O2,VNET:VA4 ->               // Transport endpoints at T2
  SIR:O2,L4:(VNI2:VAX4) ->         // On the wire
```

```
SIR:O2,VNET:VA4                  // Received at TS
```

### A.9 Scenario 6: Tenant system to Internet

Communication from a TS to the Internet assumes that the VNET for the TS is globally routable, hence no ILA translation would be needed.

If TS VA4 sends a packet to the Internet, the addresses would be:

```
VNET:VA4,Iaddr
```

### A.10 Scenario 7: Internet to tenant system

An Internet host transmits a packet to a tenant system using an externally routable tenant prefix and address. The prefix routes the packet to a gateway for the datacenter. The gateway translates the destination to an ILA address.

If a host on the Internet with address Iaddr is sending to TS VA4, the ILA translation sequence would be:

```
Iaddr,VNET:VA4 ->                  // Endpoint at Iaddr
Iaddr,L4:(VNI2:VAX4) ->            // On the wire in datacenter
Iaddr,VNET:VA4                     // Received at TS
```

### A.11 Scenario 8: IPv4 tenant system to object

A TS that is IPv4-only may communicate with an object using protocol translation. The object would be represented as an IPv4 address in the tenant’s address space, and stateless NAT64 should be usable as described in [RFC6145].

If TS VA2 communicates with object O3, the ILA translation sequence would be:

```
VA2,ADDR3 ->                       // IPv4 endpoints at TS
SIR:(VNI1:VA2),L3:O3 ->            // On the wire in datacenter
SIR:(VNI1:VA2),SIR:O3              // Received at task
```

VA2 is the IPv4 address in the tenant’s virtual network, ADDR4 is an address in the tenant’s address space that maps to the network service.

The reverse path, task sending to a TS with an IPv4 address, requires a similar protocol translation.

For object O3 communicate with TS VA2, the ILA translation sequence would be:

```
SIR:O3,SIR:(VNI1:VA2) ->          // Endpoints at T4
SIR:O3,L2:(VNI1:VA2) ->           // On the wire in datacenter
ADDR4,VA2                         // IPv4 endpoint at TS
```

A.12 Tenant to tenant system in the same virtual network

ILA may be used to allow tenants within a virtual network to communicate without the need for explicit encapsulation headers.

A.12.1 Scenario 9: TS to TS in the same VN using IPV6

If TS VA1 sends a packet to TS VA2, the ILA translation sequence would be:

```
VNET:VA1,VNET:VA2 ->              // Endpoints at VA1
VNET:VA1,L2:(VNI1,VAX2) ->        // On the wire
VNET:VA1,VNET:VA2 ->              // Received at VA2
```

A.12.2 Scenario 10: TS to TS in same VN using IPv4

For two tenant systems to communicate using IPv4 and ILA, IPv4/IPv6 protocol translation is done both on the transmit and receive.

If TS VA1 sends an IPv4 packet to TS VA2, the ILA translation sequence would be:

```
VA1,VA2 ->                        // Endpoints at VA1
SIR:(VNI1:VA1),L2:(VNI1,VA2) ->   // On the wire
VA1,VA2                           // Received at VA2
```

Note that the SIR is chosen by an ILA node as an appropriate SIR prefix in the underlay network. Tenant systems do not use SIR address for this communication, they only use virtual addresses.

A.13 Tenant system to tenant system in different virtual networks

A tenant system may be allowed to communicate with another tenant system in a different virtual network. This should only be allowed with explicit policy configuration.

A.13.1 Scenario 11: TS to TS in different VNs using IPV6

For TS VA4 to communicate with TS VA1 using IPv6 the translation sequence would be:

```
VNET2:VA4,VNET1:VA1->             // Endpoint at VA4
VNET2:VA4,L1:(VNI1,VAX1)->        // On the wire
VNET2:VA4,VNET1:VA1               // Received at VA1
```

Note that this assumes that VNET1 and VNET2 are globally routable between the two virtual networks.

### A.13.2 Scenario 12: TS to TS in different VNs using IPv4

To allow IPv4 tenant systems in different virtual networks to communicate with each other, an address representing the peer would be mapped into each tenant's address space. IPv4/IPv6 protocol translation is done on transmit and receive.

For TS VA4 to communicate with TS VA1 using IPv4 the translation sequence may be:

```
VA4,SADDR1 ->                          // IPv4 endpoint at VA4
SIR:(VNI2:VA4),L1:(VNI1,VA1)->         // On the wire
SADDR4,VA1                             // Received at VA1
```

SADDR1 is the mapped address for VA1 in VA4's address space, and SADDR4 is the mapped address for VA4 in VA1's address space.

### A.13.3 Scenario 13: IPv4 TS to IPv6 TS in different VNs

Communication may also be mixed so that an IPv4 tenant system can communicate with an IPv6 tenant system in another virtual network. IPv4/IPv6 protocol translation is done on transmit.

For TS VA4 using IPv4 to communicate with TS VA1 using IPv6 the translation sequence may be:

```
VA4,SADDR1 ->                          // IPv4 endpoint at VA4
SIR:(VNI2:VA4),L1:(VNI1,VAX1)->        // On the wire
SIR:(VNI2:VA4),VNET1:VA1               // Received at VA1
```

SADDR1 is the mapped IPv4 address for VA1 in VA4's address space.

In the reverse direction, TS VA1 using IPv6 would communicate with TS VA4 with the translation sequence:

```
VNET1:VA1,SIR:(VNI2:VA4)               // Endpoint at VA1
VNET1:VA1,L4:(VNI2:VA4)                // On the wire
SADDR1,VA4                             // Received at VA4
```

## Appendix B: unique identifier generation

The unique identifier type of ILA identifiers can address 2**60 objects. This appendix describes some method to perform allocation of identifiers for objects to avoid duplicated identifiers being allocated.

### B.1 Globally unique identifiers method

For small to moderate sized deployments the technique for creating locally assigned global identifiers described in [RFC4193] could be used. In this technique a SHA-1 digest of the time of day in NTP format and an EUI-64 identifier of the local host is performed. N bits of the result are used as the globally unique identifier.

The probability that two or more of these IDs will collide can be approximated using the formula:

```
P = 1 - exp(-N**2 / 2**(L+1))
```

where P is the probability of collision, N is the number of identifiers, and L is the length of an identifier.

The following table shows the probability of a collision for a range of identifiers using a 60-bit length.

| Identifiers | Probability of Collision |
|---|---|
| 1000 | 4.3368*10^-13 |
| 10000 | 4.3368*10^-11 |
| 100000 | 4.3368*10^-09 |
| 1000000 | 4.3368*10^-07 |

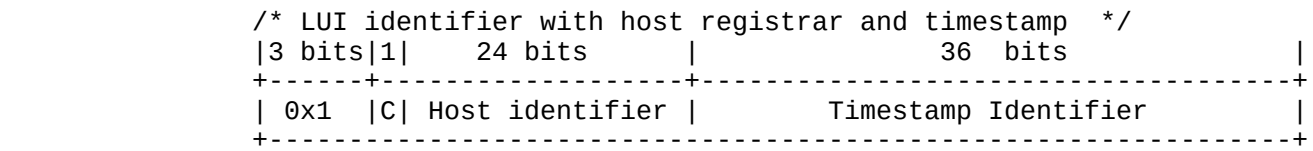
Note that locally unique identifiers may be ephemeral, for instance a task may only exist for a few seconds. This should be considered when determining the probability of identifier collision.

### B.2 Universally Unique Identifiers method

For larger deployments, hierarchical allocation may be desired. The techniques in Universally Unique Identifier (UUID) URN ([RFC4122]) can be adapted for allocating unique object identifiers in sixty bits. An identifier is split into two components: a registrar prefix and sub-identifier. The registrar prefix defines an identifier block which is managed by an agent, the sub-identifier is a unique value within the registrar block.

For instance, each host in a network could be an agent so that unique identifiers for objects could be created autonomously be the host.

The identifier might be composed of a twenty-four bit host identifier followed by a thirty-six bit timestamp. Assuming that a host can allocate up to 100 identifiers per second, this allows about 21.8 years before wrap around.




## Appendix C: Datacenter task virtualization

This section describes some details to apply ILA to virtualizing tasks in a datacenter.

### C.1 Address per task

Managing the port number space for services within a datacenter is a nontrivial problem. When a service task is created, it may run on arbitrary hosts. The typical scenario is that the task will be started on some machine and will be assigned a port number for its service. The port number must be chosen dynamically to not conflict with any other port numbers already assigned to tasks on the same machine (possibly even other instances of the same service). A canonical name for the service is entered into a database with the host address and assigned port. When a client wishes to connect to the service, it queries the database with the service name to get both the address of an instance as well as its port number. Note that DNS is not adequate for the service lookup since it does not provide port numbers.

With ILA, each service task can be assigned its own IPv6 address and therefore will logically be assigned the full port space for that address. This a dramatic simplification since each service can now use a publicly known port number that does not need to unique between services or instances. A client can perform a lookup on the service name to get an IP address of an instance and then connect to that address using a well known port number. In this case, DNS is sufficient for directing clients to instances of a service.

### C.2 Job scheduling

In the usual datacenter model, jobs are scheduled to run as tasks on some number of machines. A distributed job scheduler provides the scheduling which may entail considerable complexity since jobs will often have a variety of resource constraints. The scheduler takes these constraints into account while trying to maximize utility of

the datacenter in terms utilization, cost, latency, etc. Datacenter jobs do not typically run in virtual machines (VMs), but may run within containers. Containers are mechanisms that provide resource isolation between tasks running on the same host OS. These resources can include CPU, disk, memory, and networking.

A fundamental problem arises in that once a task for a job is scheduled on a machine, it often needs to run to completion. If the scheduler needs to schedule a higher priority job or change resource allocations, there may be little recourse but to kill tasks and restart them on a different machine. In killing a task, progress is lost which results in increased latency and wasted CPU cycles. Some tasks may checkpoint progress to minimize the amount of progress lost, but this is not a very transparent or general solution.

An alternative approach is to allow transparent job migration. The scheduler may migrate running jobs from one machine to another.

### C.3 Task migration

Under the orchestration of the job scheduler, the steps to migrate a job may be:

1) Stop running tasks for the job.
2) Package the runtime state of the job. The runtime state is derived from the containers for the jobs.
3) Send the runtime state of the job to the new machine where the job is to run.
4) Instantiate the job’s state on the new machine.
5) Start the tasks for the job continuing from the point at which it was stopped.

This model similar to virtual machine (VM) migration except that the runtime state is typically much less data-- just task state as opposed to a full OS image. Task state may be compressed to reduce latency in migration.

#### C.3.1 Address migration

ILA facilitates address (specifically SIR address) migration between hosts as part of task migration or for other purposes. The steps in migrating an address might be:

1) Configure address on the target host.

2) Suspend use of the address on the old host. This includes handling established connections (see next section). A state may be established to drop packets or send ICMP destination

unreachable when packets to the migrated address are received.

3) Update the identifier to locator mapping database. Depending on the control plane implementation this may include pushing the new mapping to hosts.

4) Communicating hosts will learn of the new mapping via a control plane either by participation in a protocol for mapping propagation or by the ILA resolution protocol.

C.3.2 Connection migration

When a task and its addresses are migrated between machines, the disposition of existing TCP connections needs to be considered.

The simplest course of action is to drop TCP connections across a migration. Since migrations should be relatively rare events, it is conceivable that TCP connections could be automatically closed in the network stack during a migration event. If the applications running are known to handle this gracefully (i.e. reopen dropped connections) then this may be viable.

For seamless migration, open connections may be migrated between hosts. Migration of these entails pausing the connection, packaging connection state and sending to target, instantiating connection state in the peer stack, and restarting the connection. From the time the connection is paused to the time it is running again in the new stack, packets received for the connection should be silently dropped. For some period of time, the old stack will need to keep a record of the migrated connection. If it receives a packet, it should either silently drop the packet or forward it to the new location.

Author’s Address

Tom Herbert
Quantonium
4701 Patrick Henry
Santa Clara, CA
EMail: tom@herbertland.com

Petr Lapukhov
1 Hacker Way
Menlo Parck, CA
EMail: petr@fb.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2017

L. Lhotka
CZ.NIC
A. Lindem
Cisco Systems

November 03, 2016

# A YANG Data Model for Routing Management
# draft-ietf-netmod-routing-cfg-25

## Abstract

This document contains a specification of three YANG modules and one submodule. Together they form the core routing data model which serves as a framework for configuring and managing a routing subsystem. It is expected that these modules will be augmented by additional YANG modules defining data models for control plane protocols, route filters and other functions. The core routing data model provides common building blocks for such extensions -- routes, routing information bases (RIB), and control plane protocols.

## 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 May 7, 2017.

## Copyright Notice

Table of Contents

## 1. Introduction

This document contains a specification of the following YANG modules:

- Module "ietf-routing" provides generic components of a routing data model.

- Module "ietf-ipv4-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv4 unicast.

- Module "ietf-ipv6-unicast-routing" augments the "ietf-routing" module with additional data specific to IPv6 unicast. Its submodule "ietf-ipv6-router-advertisements" also augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules with IPv6 router configuration variables required by [RFC4861].

These modules together define the so-called core routing data model, which is intended as a basis for future data model development covering more sophisticated routing systems. While these three modules can be directly used for simple IP devices with static routing (see Appendix B), their main purpose is to provide essential building blocks for more complicated data models involving multiple control plane protocols, multicast routing, additional address families, and advanced functions such as route filtering or policy routing. To this end, it is expected that the core routing data model will be augmented by numerous modules developed by other IETF working groups.

## 2. Terminology and 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].

The following terms are defined in [RFC6241]:

- client,
- message,
- protocol operation,
- server.

The following terms are defined in [RFC7950]:

- action,
- augment,
- configuration data,
- container,
- container with presence,
- data model,
- data node,
- feature,
- leaf,
- list,
- mandatory node,
- module,
- schema tree,
- state data,
- RPC operation.

### 2.1. Glossary of New Terms

core routing data model: YANG data model comprising "ietf-routing", "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing" modules.

direct route: a route to a directly connected network.

routing information base (RIB): An object containing a list of routes together with other information. See Section 5.2 for details.

system-controlled entry: An entry of a list in state data ("config false") that is created by the system independently of what has been explicitly configured. See Section 4.1 for details.

user-controlled entry: An entry of a list in state data ("config false") that is created and deleted as a direct consequence of certain configuration changes. See Section 4.1 for details.

### 2.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Appendix A, and similar diagrams of its various subtrees appear in the main text.

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

### 2.3. Prefixes in Data Node Names

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

| Prefix | YANG module | Reference |
|---|---|---|
| if | ietf-interfaces | [RFC7223] |
| ip | ietf-ip | [RFC7277] |
| rt | ietf-routing | Section 7 |
| v4ur | ietf-ipv4-unicast-routing | Section 8 |
| v6ur | ietf-ipv6-unicast-routing | Section 9 |
| yang | ietf-yang-types | [RFC6991] |
| inet | ietf-inet-types | [RFC6991] |

Table 1: Prefixes and corresponding YANG modules

## 3. Objectives

The initial design of the core routing data model was driven by the following objectives:

- The data model should be suitable for the common address families, in particular IPv4 and IPv6, and for unicast and multicast routing, as well as Multiprotocol Label Switching (MPLS).

- A simple IP routing system, such as one that uses only static routing, should be configurable in a simple way, ideally without any need to develop additional YANG modules.

- On the other hand, the core routing framework must allow for complicated implementations involving multiple routing information bases (RIB) and multiple control plane protocols, as well as controlled redistributions of routing information.

- Device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces. Therefore, the framework should be flexible enough to facilitate such a mapping and accommodate data models with different logic.

## 4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules and one submodule. The first module, "ietf-routing", defines the generic components of a routing system. The other two modules, "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing", augment the "ietf-routing" module with additional data nodes that are needed for IPv4 and IPv6 unicast routing, respectively. Module "ietf-ipv6-unicast-routing" has a submodule, "ietf-ipv6-router-advertisements", that augments the "ietf-interfaces" [RFC7223] and "ietf-ip" [RFC7277] modules with configuration variables for IPv6 router advertisements as required by [RFC4861]. Figures 1 and 2 show abridged views of the configuration and state data hierarchies. See Appendix A for the complete data trees.

```
+--rw routing
   +--rw router-id?
   +--rw control-plane-protocols
   |  +--rw control-plane-protocol* [type name]
   |     +--rw type
   |     +--rw name
   |     +--rw description?
   |     +--rw static-routes
   |        +--rw v6ur:ipv6
   |        |     ...
   |        +--rw v4ur:ipv4
   |              ...
   +--rw ribs
      +--rw rib* [name]
         +--rw name
         +--rw address-family?
         +--rw description?
```

Figure 1: Configuration data hierarchy.

```
+--ro routing-state
   +--ro router-id?
   +--ro interfaces
   |  +--ro interface*
   +--ro control-plane-protocols
   |  +--ro control-plane-protocol* [type name]
   |     +--ro type
   |     +--ro name
   +--ro ribs
      +--ro rib* [name]
         +--ro name
         +--ro address-family
         +--ro default-rib?
         +--ro routes
         |  +--ro route*
         |        ...
```

Figure 2: State data hierarchy.

As can be seen from Figures 1 and 2, the core routing data model introduces several generic components of a routing framework: routes, RIBs containing lists of routes, and control plane protocols. Section 5 describes these components in more detail.

### 4.1. System-Controlled and User-Controlled List Entries

The core routing data model defines several lists in the schema tree, such as "rib", that have to be populated with at least one entry in any properly functioning device, and additional entries may be configured by a client.

In such a list, the server creates the required item as a so-called system-controlled entry in state data, i.e., inside the "routing-state" container.

An example can be seen in Appendix D: the "/routing-state/ribs/rib" list has two system-controlled entries named "ipv4-master" and "ipv6-master".

Additional entries may be created in the configuration by a client, e.g., via the NETCONF protocol. These are so-called user-controlled entries. If the server accepts a configured user-controlled entry, then this entry also appears in the state data version of the list.

Corresponding entries in both versions of the list (in state data and configuration) have the same value of the list key.

A client may also provide supplemental configuration of system-controlled entries. To do so, the client creates a new entry in the configuration with the desired contents. In order to bind this entry to the corresponding entry in the state data list, the key of the configuration entry has to be set to the same value as the key of the state entry.

Deleting a user-controlled entry from the configuration list results in the removal of the corresponding entry in the state data list. In contrast, if a system-controlled entry is deleted from the configuration list, only the extra configuration specified in that entry is removed but the corresponding state data entry remains in the list.

## 5. Basic Building Blocks

This section describes the essential components of the core routing data model.

### 5.1. Route

Routes are basic elements of information in a routing system. The core routing data model defines only the following minimal set of route attributes:

- "destination-prefix": address prefix specifying the set of destination addresses for which the route may be used. This attribute is mandatory.

- "route-preference": an integer value (also known as administrative distance) that is used for selecting a preferred route among routes with the same destination prefix. A lower value means a more preferred route.

- "next-hop": determines the outgoing interface and/or next-hop address(es), other operation to be performed with a packet.

Routes are primarily state data that appear as entries of RIBs (Section 5.2) but they may also be found in configuration data, for example as manually configured static routes. In the latter case, configurable route attributes are generally a subset of attributes defined for RIB routes.

### 5.2. Routing Information Base (RIB)

Every implementation of the core routing data model manages one or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Each RIB contains only routes

of one address family. An address family is represented by an identity derived from the "rt:address-family" base identity.

In the core routing data model, RIBs are state data represented as entries of the list "/routing-state/ribs/rib". The contents of RIBs are controlled and manipulated by control plane protocol operations which may result in route additions, removals and modifications. This also includes manipulations via the "static" and/or "direct" pseudo-protocols, see Section 5.3.1.

For every supported address family, exactly one RIB MUST be marked as the so-called default RIB. Its role is explained in Section 5.3.

Simple router implementations that do not advertise the feature "multiple-ribs" will typically create one system-controlled RIB per supported address family, and mark it as the default RIB.

More complex router implementations advertising the "multiple-ribs" feature support multiple RIBs per address family that can be used for policy routing and other purposes.

The following action (see Section 7.15 of [RFC7950]) is defined for the "rib" list:

- active-route -- return the active RIB route for the destination address that is specified as the action’s input parameter.

## 5.3. Control Plane Protocol

The core routing data model provides an open-ended framework for defining multiple control plane protocol instances, e.g., for Layer 3 routing protocols. Each control plane protocol instance MUST be assigned a type, which is an identity derived from the "rt:control-plane-protocol" base identity. The core routing data model defines two identities for the direct and static pseudo-protocols (Section 5.3.1).

Multiple control plane protocol instances of the same type MAY be configured.

### 5.3.1. Routing Pseudo-Protocols

The core routing data model defines two special routing protocol types -- "direct" and "static". Both are in fact pseudo-protocols, which means that they are confined to the local device and do not exchange any routing information with adjacent routers.

Every implementation of the core routing data model MUST provide exactly one instance of the "direct" pseudo-protocol type. It is the source of direct routes for all configured address families. Direct routes are normally supplied by the operating system kernel, based on the configuration of network interface addresses, see Section 6.2.

A pseudo-protocol of the type "static" allows for specifying routes manually. It MAY be configured in zero or multiple instances, although a typical configuration will have exactly one instance.

### 5.3.2. Defining New Control Plane Protocols

It is expected that future YANG modules will create data models for additional control plane protocol types. Such a new module has to define the protocol-specific configuration and state data, and it has to integrate it into the core routing framework in the following way:

- A new identity MUST be defined for the control plane protocol and its base identity MUST be set to "rt:control-plane-protocol", or to an identity derived from "rt:control-plane-protocol".

- Additional route attributes MAY be defined, preferably in one place by means of defining a YANG grouping. The new attributes have to be inserted by augmenting the definitions of the nodes

  /rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route

  and

  /rt:routing-state/rt:ribs/rt:rib/rt:output/rt:route,

  and possibly other places in the configuration, state data, notifications, and input/output parameters of actions or RPC operations.

- Configuration parameters and/or state data for the new protocol can be defined by augmenting the "control-plane-protocol" data node under both "/routing" and "/routing-state".

By using a "when" statement, the augmented configuration parameters and state data specific to the new protocol SHOULD be made conditional and valid only if the value of "rt:type" or "rt:source-protocol" is equal to (or derived from) the new protocol's identity.

It is also RECOMMENDED that protocol-specific data nodes be encapsulated in an appropriately named container with presence. Such a container may contain mandatory data nodes that are otherwise forbidden at the top level of an augment.

The above steps are implemented by the example YANG module for the RIP routing protocol in Appendix C.

### 5.4. Parameters of IPv6 Router Advertisements

YANG module "ietf-ipv6-router-advertisements" (Section 9.1), which is a submodule of the "ietf-ipv6-unicast-routing" module, augments the configuration and state data of IPv6 interfaces with definitions of the following variables as required by [RFC4861], sec. 6.2.1:

- send-advertisements,
- max-rtr-adv-interval,
- min-rtr-adv-interval,
- managed-flag,
- other-config-flag,
- link-mtu,
- reachable-time,
- retrans-timer,
- cur-hop-limit,
- default-lifetime,
- prefix-list: a list of prefixes to be advertised.

  The following parameters are associated with each prefix in the list:

  * valid-lifetime,
  * on-link-flag,
  * preferred-lifetime,
  * autonomous-flag.

NOTES:

1. The "IsRouter" flag, which is also required by [RFC4861], is implemented in the "ietf-ip" module [RFC7277] (leaf "ip:forwarding").

2. The original specification [RFC4861] allows the implementations to decide whether the "valid-lifetime" and "preferred-lifetime" parameters remain the same in consecutive advertisements, or decrement in real time. However, the latter behavior seems problematic because the values might be reset again to the (higher) configured values after a configuration is reloaded. Moreover, no implementation is known to use the decrementing behavior. The "ietf-ipv6-router-advertisements" submodule therefore stipulates the former behavior with constant values.

# 6. Interactions with Other YANG Modules

The semantics of the core routing data model also depends on several configuration parameters that are defined in other YANG modules.

## 6.1. Module "ietf-interfaces"

The following boolean switch is defined in the "ietf-interfaces" YANG module [RFC7223]:

/if:interfaces/if:interface/if:enabled

If this switch is set to "false" for a network layer interface, then all routing and forwarding functions MUST be disabled on that interface.

## 6.2. Module "ietf-ip"

The following boolean switches are defined in the "ietf-ip" YANG module [RFC7277]:

/if:interfaces/if:interface/ip:ipv4/ip:enabled

If this switch is set to "false" for a network layer interface, then all IPv4 routing and forwarding functions MUST be disabled on that interface.

/if:interfaces/if:interface/ip:ipv4/ip:forwarding

If this switch is set to "false" for a network layer interface, then the forwarding of IPv4 datagrams through this interface MUST be disabled. However, the interface MAY participate in other IPv4 routing functions, such as routing protocols.

/if:interfaces/if:interface/ip:ipv6/ip:enabled

If this switch is set to "false" for a network layer interface, then all IPv6 routing and forwarding functions MUST be disabled on that interface.

/if:interfaces/if:interface/ip:ipv6/ip:forwarding

If this switch is set to "false" for a network layer interface, then the forwarding of IPv6 datagrams through this interface MUST be disabled. However, the interface MAY participate in other IPv6 routing functions, such as routing protocols.

In addition, the "ietf-ip" module allows for configuring IPv4 and IPv6 addresses and network prefixes or masks on network layer interfaces. Configuration of these parameters on an enabled interface MUST result in an immediate creation of the corresponding direct route. The destination prefix of this route is set according to the configured IP address and network prefix/mask, and the interface is set as the outgoing interface for that route.

## 7. Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-routing@2016-11-03.yang"

```
module ietf-routing {

  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-routing";

  prefix "rt";

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

  import ietf-interfaces {
    prefix "if";
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:   <https://tools.ietf.org/wg/netmod/>
```

```
        WG List:  <mailto:netmod@ietf.org>

        WG Chair: Lou Berger
                  <mailto:lberger@labn.net>

        WG Chair: Kent Watsen
                  <mailto:kwatsen@juniper.net>

        Editor:   Ladislav Lhotka
                  <mailto:lhotka@nic.cz>

        Editor:   Acee Lindem
                  <mailto:acee@cisco.com>";

    description
      "This YANG module defines essential components for the management
       of a routing subsystem.

       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
       (https://trustee.ietf.org/license-info).

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
       NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
       'OPTIONAL' in the module text are to be interpreted as described
       in RFC 2119 (https://tools.ietf.org/html/rfc2119).

       This version of this YANG module is part of RFC XXXX
       (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
       full legal notices.";

    revision 2016-11-03 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: A YANG Data Model for Routing Management";
    }

    /* Features */

    feature multiple-ribs {
      description
```

```
        "This feature indicates that the server supports user-defined
         RIBs.

         Servers that do not advertise this feature SHOULD provide
         exactly one system-controlled RIB per supported address family
         and make them also the default RIBs. These RIBs then appear as
         entries of the list /routing-state/ribs/rib.";
    }

    feature router-id {
      description
        "This feature indicates that the server supports configuration
         of an explicit 32-bit router ID that is used by some routing
         protocols.

         Servers that do not advertise this feature set a router ID
         algorithmically, usually to one of configured IPv4 addresses.
         However, this algorithm is implementation-specific.";
    }

    /* Identities */

    identity address-family {
      description
        "Base identity from which identities describing address
         families are derived.";
    }

    identity ipv4 {
      base address-family;
      description
        "This identity represents IPv4 address family.";
    }

    identity ipv6 {
      base address-family;
      description
        "This identity represents IPv6 address family.";
    }

    identity control-plane-protocol {
      description
        "Base identity from which control plane protocol identities are
         derived.";
    }

    identity routing-protocol {
      base control-plane-protocol;
```

```
    description
      "Identity from which Layer 3 routing protocol identities are
       derived.";
  }

  identity direct {
    base routing-protocol;
    description
      "Routing pseudo-protocol that provides routes to directly
       connected networks.";
  }

  identity static {
    base routing-protocol;
    description
      "Static routing pseudo-protocol.";
  }

  /* Type Definitions */

  typedef route-preference {
    type uint32;
    description
      "This type is used for route preferences.";
  }

  /* Groupings */

  grouping address-family {
    description
      "This grouping provides a leaf identifying an address
       family.";
    leaf address-family {
      type identityref {
        base address-family;
      }
      mandatory "true";
      description
        "Address family.";
    }
  }

  grouping router-id {
    description
      "This grouping provides router ID.";
    leaf router-id {
      type yang:dotted-quad;
      description
```

```
          "A 32-bit number in the form of a dotted quad that is used by
           some routing protocols identifying a router.";
        reference
          "RFC 2328: OSPF Version 2.";
      }
    }

    grouping special-next-hop {
      description
        "This grouping provides a leaf with an enumeration of special
         next-hops.";
      leaf special-next-hop {
        type enumeration {
          enum blackhole {
            description
              "Silently discard the packet.";
          }
          enum unreachable {
            description
              "Discard the packet and notify the sender with an error
               message indicating that the destination host is
               unreachable.";
          }
          enum prohibit {
            description
              "Discard the packet and notify the sender with an error
               message indicating that the communication is
               administratively prohibited.";
          }
          enum receive {
            description
              "The packet will be received by the local system.";
          }
        }
        description
          "Special next-hop options.";
      }
    }

    grouping next-hop-content {
      description
        "Generic parameters of next-hops in static routes.";
      choice next-hop-options {
        mandatory "true";
        description
          "Options for next-hops in static routes.

           It is expected that further cases will be added through
```

```
           augments from other modules.";
        case simple-next-hop {
          description
            "This case represents a simple next hop consisting of the
             next-hop address and/or outgoing interface.

             Modules for address families MUST augment this case with a
             leaf containing a next-hop address of that address
             family.";
          leaf outgoing-interface {
            type if:interface-ref;
            description
              "Name of the outgoing interface.";
          }
        }
        case special-next-hop {
          uses special-next-hop;
        }
        case next-hop-list {
          container next-hop-list {
            description
              "Container for multiple next-hops.";
            list next-hop {
              key "index";
              description
                "An entry of a next-hop list.

                 Modules for address families MUST augment this list
                 with a leaf containing a next-hop address of that
                 address family.";
              leaf index {
                type string;
                description
                  "An user-specified identifier utilised to uniquely
                   reference the next-hop entry in the next-hop list.
                   The value of this index has no semantic meaning
                   other than for referencing the entry.";
              }
              leaf outgoing-interface {
                type if:interface-ref;
                description
                  "Name of the outgoing interface.";
              }
            }
          }
        }
      }
    }
```

```
     grouping next-hop-state-content {
       description
         "Generic parameters of next-hops in state data.";
       choice next-hop-options {
         mandatory "true";
         description
           "Options for next-hops in state data.

            It is expected that further cases will be added through
            augments from other modules, e.g., for recursive
            next-hops.";
         case simple-next-hop {
           description
             "This case represents a simple next hop consisting of the
              next-hop address and/or outgoing interface.

              Modules for address families MUST augment this case with a
              leaf containing a next-hop address of that address
              family.";
           leaf outgoing-interface {
             type if:interface-state-ref;
             description
               "Name of the outgoing interface.";
           }
         }
         case special-next-hop {
           uses special-next-hop;
         }
         case next-hop-list {
           container next-hop-list {
             description
               "Container for multiple next-hops.";
             list next-hop {
               description
                 "An entry of a next-hop list.

                  Modules for address families MUST augment this list
                  with a leaf containing a next-hop address of that
                  address family.";
               leaf outgoing-interface {
                 type if:interface-state-ref;
                 description
                   "Name of the outgoing interface.";
               }
             }
           }
         }
       }
```

```
  }

  grouping route-metadata {
    description
      "Common route metadata.";
    leaf source-protocol {
      type identityref {
        base routing-protocol;
      }
      mandatory "true";
      description
        "Type of the routing protocol from which the route
         originated.";
    }
    leaf active {
      type empty;
      description
        "Presence of this leaf indicates that the route is preferred
         among all routes in the same RIB that have the same
         destination prefix.";
    }
    leaf last-updated {
      type yang:date-and-time;
      description
        "Time stamp of the last modification of the route. If the
         route was never modified, it is the time when the route was
         inserted into the RIB.";
    }
  }

  /* State data */

  container routing-state {
    config "false";
    description
      "State data of the routing subsystem.";
    uses router-id {
      description
        "Global router ID.

         It may be either configured or assigned algorithmically by
         the implementation.";
    }
    container interfaces {
      description
        "Network layer interfaces used for routing.";
      leaf-list interface {
        type if:interface-state-ref;
```

```
          description
            "Each entry is a reference to the name of a configured
             network layer interface.";
        }
      }
      container control-plane-protocols {
        description
          "Container for the list of routing protocol instances.";
        list control-plane-protocol {
          key "type name";
          description
            "State data of a control plane protocol instance.

             An implementation MUST provide exactly one
             system-controlled instance of the 'direct'
             pseudo-protocol. Instances of other control plane
             protocols MAY be created by configuration.";
          leaf type {
            type identityref {
              base control-plane-protocol;
            }
            description
              "Type of the control plane protocol.";
          }
          leaf name {
            type string;
            description
              "The name of the control plane protocol instance.

               For system-controlled instances this name is persistent,
               i.e., it SHOULD NOT change across reboots.";
          }
        }
      }
      container ribs {
        description
          "Container for RIBs.";
        list rib {
          key "name";
          min-elements "1";
          description
            "Each entry represents a RIB identified by the 'name' key.
             All routes in a RIB MUST belong to the same address
             family.

             An implementation SHOULD provide one system-controlled
             default RIB for each supported address family.";
          leaf name {
```

```
            type string;
            description
              "The name of the RIB.";
          }
          uses address-family;
          leaf default-rib {
            if-feature "multiple-ribs";
            type boolean;
            default "true";
            description
              "This flag has the value of 'true' if and only if the RIB
               is the default RIB for the given address family.

               By default, control plane protocols place their routes
               in the default RIBs.";
          }
          container routes {
            description
              "Current content of the RIB.";
            list route {
              description
                "A RIB route entry. This data node MUST be augmented
                 with information specific for routes of each address
                 family.";
              leaf route-preference {
                type route-preference;
                description
                  "This route attribute, also known as administrative
                   distance, allows for selecting the preferred route
                   among routes with the same destination prefix. A
                   smaller value means a more preferred route.";
              }
              container next-hop {
                description
                  "Route's next-hop attribute.";
                uses next-hop-state-content;
              }
              uses route-metadata;
            }
          }
          action active-route {
            description
              "Return the active RIB route that is used for the
               destination address.

               Address family specific modules MUST augment input
               parameters with a leaf named 'destination-address'.";
            output {
```

```
            container route {
              description
                "The active RIB route for the specified destination.

                 If no route exists in the RIB for the destination
                 address, no output is returned.

                 Address family specific modules MUST augment this
                 container with appropriate route contents.";
              container next-hop {
                description
                  "Route's next-hop attribute.";
                uses next-hop-state-content;
              }
              uses route-metadata;
            }
          }
        }
      }
    }

    /* Configuration Data */

    container routing {
      description
        "Configuration parameters for the routing subsystem.";
      uses router-id {
        if-feature "router-id";
        description
          "Configuration of the global router ID. Routing protocols
           that use router ID can use this parameter or override it
           with another value.";
      }
      container control-plane-protocols {
        description
          "Configuration of control plane protocol instances.";
        list control-plane-protocol {
          key "type name";
          description
            "Each entry contains configuration of a control plane
             protocol instance.";
          leaf type {
            type identityref {
              base control-plane-protocol;
            }
            description
              "Type of the control plane protocol - an identity derived
```

```
            from the 'control-plane-protocol' base identity.";
        }
        leaf name {
          type string;
          description
            "An arbitrary name of the control plane protocol
             instance.";
        }
        leaf description {
          type string;
          description
            "Textual description of the control plane protocol
             instance.";
        }
        container static-routes {
          when "derived-from-or-self(../type, 'rt:static')" {
            description
              "This container is only valid for the 'static' routing
               protocol.";
          }
          description
            "Configuration of the 'static' pseudo-protocol.

             Address-family-specific modules augment this node with
             their lists of routes.";
        }
      }
    }
    container ribs {
      description
        "Configuration of RIBs.";
      list rib {
        key "name";
        description
          "Each entry contains configuration for a RIB identified by
           the 'name' key.

           Entries having the same key as a system-controlled entry
           of the list /routing-state/ribs/rib are used for
           configuring parameters of that entry. Other entries define
           additional user-controlled RIBs.";
        leaf name {
          type string;
          description
            "The name of the RIB.

             For system-controlled entries, the value of this leaf
             must be the same as the name of the corresponding entry
```

```
                in state data.

                For user-controlled entries, an arbitrary name can be
                used.";
            }
            uses address-family {
              description
                "Address family of the RIB.

                It is mandatory for user-controlled RIBs. For
                system-controlled RIBs it can be omitted, otherwise it
                must match the address family of the corresponding state
                entry.";
              refine "address-family" {
                mandatory "false";
              }
            }
            leaf description {
              type string;
              description
                "Textual description of the RIB.";
            }
          }
        }
      }
    }
```

<CODE ENDS>

## 8. IPv4 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-ipv4-unicast-routing@2016-11-03.yang"

```
module ietf-ipv4-unicast-routing {

  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing";

  prefix "v4ur";

  import ietf-routing {
    prefix "rt";
  }
```

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

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:   <https://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     WG Chair: Lou Berger
               <mailto:lberger@labn.net>

     WG Chair: Kent Watsen
               <mailto:kwatsen@juniper.net>

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>

     Editor:   Acee Lindem
               <mailto:acee@cisco.com>";

  description
    "This YANG module augments the 'ietf-routing' module with basic
     configuration and state data for IPv4 unicast routing.

     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
     (https://trustee.ietf.org/license-info).

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
     'OPTIONAL' in the module text are to be interpreted as described
     in RFC 2119 (https://tools.ietf.org/html/rfc2119).

     This version of this YANG module is part of RFC XXXX
     (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
     full legal notices.";

  revision 2016-11-03 {
    description
```

```
        "Initial revision.";
      reference
        "RFC XXXX: A YANG Data Model for Routing Management";
    }

    /* Identities */

    identity ipv4-unicast {
      base rt:ipv4;
      description
        "This identity represents the IPv4 unicast address family.";
    }

    /* State data */

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
      when "derived-from-or-self(../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "This leaf augments an IPv4 unicast route.";
      leaf destination-prefix {
        type inet:ipv4-prefix;
        description
          "IPv4 destination prefix.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in IPv4 unicast routes.";
      leaf next-hop-address {
        type inet:ipv4-address;
        description
          "IPv4 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:next-hop-list/"
```

```
          + "rt:next-hop-list/rt:next-hop" {
        when "derived-from-or-self(../../../../../rt:address-family, "
           + "'v4ur:ipv4-unicast')" {
          description
            "This augment is valid only for IPv4 unicast.";
        }
        description
          "This leaf augments the 'next-hop-list' case of IPv4 unicast
           routes.";
        leaf address {
          type inet:ipv4-address;
          description
            "IPv4 address of the next-hop.";
        }
      }

      augment
        "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
        when "derived-from-or-self(../rt:address-family, "
           + "'v4ur:ipv4-unicast')" {
          description
            "This augment is valid only for IPv4 unicast RIBs.";
        }
        description
          "This augment adds the input parameter of the 'active-route'
           action.";
        leaf destination-address {
          type inet:ipv4-address;
          description
            "IPv4 destination address.";
        }
      }

      augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
            + "rt:output/rt:route" {
        when "derived-from-or-self(../../rt:address-family, "
           + "'v4ur:ipv4-unicast')" {
          description
            "This augment is valid only for IPv4 unicast.";
        }
        description
          "This augment adds the destination prefix to the reply of the
           'active-route' action.";
        leaf destination-prefix {
          type inet:ipv4-prefix;
          description
            "IPv4 destination prefix.";
        }
```

```
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv4-address;
        description
          "IPv4 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
      when "derived-from-or-self(../../../../../rt:address-family, "
         + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      description
        "Augment 'next-hop-list' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv4-address;
        description
          "IPv4 address of the next-hop.";
      }
    }

    /* Configuration data */

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes" {
      description
        "This augment defines the configuration of the 'static'
         pseudo-protocol with data specific to IPv4 unicast.";
      container ipv4 {
        description
          "Configuration of a 'static' pseudo-protocol instance
```

```
           consists of a list of routes.";
         list route {
           key "destination-prefix";
           description
             "A list of static routes.";
           leaf destination-prefix {
             type inet:ipv4-prefix;
             mandatory "true";
             description
               "IPv4 destination prefix.";
           }
           leaf description {
             type string;
             description
               "Textual description of the route.";
           }
           container next-hop {
             description
               "Configuration of next-hop.";
             uses rt:next-hop-content {
               augment "next-hop-options/simple-next-hop" {
                 description
                   "Augment 'simple-next-hop' case in IPv4 static
                    routes.";
                 leaf next-hop-address {
                   type inet:ipv4-address;
                   description
                     "IPv4 address of the next-hop.";
                 }
               }
               augment "next-hop-options/next-hop-list/next-hop-list/"
                     + "next-hop" {
                 description
                   "Augment 'next-hop-list' case in IPv4 static
                    routes.";
                 leaf next-hop-address {
                   type inet:ipv4-address;
                   description
                     "IPv4 address of the next-hop.";
                 }
               }
             }
           }
         }
       }
     }
   }
```

<CODE ENDS>

## 9. IPv6 Unicast Routing Management YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-ipv6-unicast-routing@2016-11-03.yang"

```
module ietf-ipv6-unicast-routing {

  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing";

  prefix "v6ur";

  import ietf-routing {
    prefix "rt";
  }

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

  include ietf-ipv6-router-advertisements {
    revision-date 2016-11-03;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

  contact
    "WG Web:   <https://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     WG Chair: Lou Berger
               <mailto:lberger@labn.net>

     WG Chair: Kent Watsen
               <mailto:kwatsen@juniper.net>

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>

     Editor:   Acee Lindem
               <mailto:acee@cisco.com>";
```

```
     description
       "This YANG module augments the 'ietf-routing' module with basic
        configuration and state data for IPv6 unicast routing.

        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
        (https://trustee.ietf.org/license-info).

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
        'OPTIONAL' in the module text are to be interpreted as described
        in RFC 2119 (https://tools.ietf.org/html/rfc2119).

        This version of this YANG module is part of RFC XXXX
        (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
        full legal notices.";

     revision 2016-11-03 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management";
     }

     /* Identities */

     identity ipv6-unicast {
       base rt:ipv6;
       description
         "This identity represents the IPv6 unicast address family.";
     }

     /* State data */

     augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
       when "derived-from-or-self(../../rt:address-family, "
          + "'v6ur:ipv6-unicast')" {
         description
           "This augment is valid only for IPv6 unicast.";
       }
       description
         "This leaf augments an IPv6 unicast route.";
```

```
      leaf destination-prefix {
        type inet:ipv6-prefix;
        description
          "IPv6 destination prefix.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in IPv6 unicast routes.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:next-hop-list/"
          + "rt:next-hop-list/rt:next-hop" {
      when "derived-from-or-self(../../../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "This leaf augments the 'next-hop-list' case of IPv6 unicast
         routes.";
      leaf address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    augment
      "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/rt:input" {
      when "derived-from-or-self(../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast RIBs.";
      }
```

```
      description
        "This augment adds the input parameter of the 'active-route'
         action.";
      leaf destination-address {
        type inet:ipv6-address;
        description
          "IPv6 destination address.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route" {
      when "derived-from-or-self(../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "This augment adds the destination prefix to the reply of the
         'active-route' action.";
      leaf destination-prefix {
        type inet:ipv6-prefix;
        description
          "IPv6 destination prefix.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:simple-next-hop" {
      when "derived-from-or-self(../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "Augment 'simple-next-hop' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route/rt:next-hop/rt:next-hop-options/"
          + "rt:next-hop-list/rt:next-hop-list/rt:next-hop" {
```

```
      when "derived-from-or-self(../../../../../rt:address-family, "
         + "'v6ur:ipv6-unicast')" {
        description
          "This augment is valid only for IPv6 unicast.";
      }
      description
        "Augment 'next-hop-list' case in the reply to the
         'active-route' action.";
      leaf next-hop-address {
        type inet:ipv6-address;
        description
          "IPv6 address of the next-hop.";
      }
    }

    /* Configuration data */

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol/rt:static-routes" {
      description
        "This augment defines the configuration of the 'static'
         pseudo-protocol with data specific to IPv6 unicast.";
      container ipv6 {
        description
          "Configuration of a 'static' pseudo-protocol instance
           consists of a list of routes.";
        list route {
          key "destination-prefix";
          description
            "A list of static routes.";
          leaf destination-prefix {
            type inet:ipv6-prefix;
            mandatory "true";
            description
              "IPv6 destination prefix.";
          }
          leaf description {
            type string;
            description
              "Textual description of the route.";
          }
          container next-hop {
            description
              "Configuration of next-hop.";
            uses rt:next-hop-content {
              augment "next-hop-options/simple-next-hop" {
                description
                  "Augment 'simple-next-hop' case in IPv6 static
```

```
                   routes.";
                 leaf next-hop-address {
                   type inet:ipv6-address;
                   description
                     "IPv6 address of the next-hop.";
                 }
               }
               augment "next-hop-options/next-hop-list/next-hop-list/"
                     + "next-hop" {
                 description
                   "Augment 'next-hop-list' case in IPv6 static
                    routes.";
                 leaf next-hop-address {
                   type inet:ipv6-address;
                   description
                     "IPv6 address of the next-hop.";
                 }
               }
             }
           }
         }
       }
     }
   }
```

<CODE ENDS>

## 9.1. IPv6 Router Advertisements Submodule

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

<CODE BEGINS> file "ietf-ipv6-router-advertisements@2016-11-03.yang"

```
submodule ietf-ipv6-router-advertisements {

  yang-version "1.1";

  belongs-to ietf-ipv6-unicast-routing {
    prefix "v6ur";
  }

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

  import ietf-interfaces {
```

```
      prefix "if";
    }

    import ietf-ip {
      prefix "ip";
    }

    organization
      "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

    contact
      "WG Web:   <https://tools.ietf.org/wg/netmod/>
       WG List:  <mailto:netmod@ietf.org>

       WG Chair: Lou Berger
                 <mailto:lberger@labn.net>

       WG Chair: Kent Watsen
                 <mailto:kwatsen@juniper.net>

       Editor:   Ladislav Lhotka
                 <mailto:lhotka@nic.cz>

       Editor:   Acee Lindem
                 <mailto:acee@cisco.com>";

    description
      "This YANG module augments the 'ietf-ip' module with
       configuration and state data of IPv6 router advertisements.

       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
       (https://trustee.ietf.org/license-info).

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
       NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
       'OPTIONAL' in the module text are to be interpreted as described
       in RFC 2119 (https://tools.ietf.org/html/rfc2119).

       This version of this YANG module is part of RFC XXXX
       (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
       full legal notices.";
```

```
     reference
       "RFC 4861: Neighbor Discovery for IP version 6 (IPv6).";

     revision 2016-11-03 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management";
     }

     /* State data */

     augment "/if:interfaces-state/if:interface/ip:ipv6" {
       description
         "Augment interface state data with parameters of IPv6 router
          advertisements.";
       container ipv6-router-advertisements {
         description
           "Parameters of IPv6 Router Advertisements.";
         leaf send-advertisements {
           type boolean;
           description
             "A flag indicating whether or not the router sends periodic
              Router Advertisements and responds to Router
              Solicitations.";
         }
         leaf max-rtr-adv-interval {
           type uint16 {
             range "4..1800";
           }
           units "seconds";
           description
             "The maximum time allowed between sending unsolicited
              multicast Router Advertisements from the interface.";
         }
         leaf min-rtr-adv-interval {
           type uint16 {
             range "3..1350";
           }
           units "seconds";
           description
             "The minimum time allowed between sending unsolicited
              multicast Router Advertisements from the interface.";
         }
         leaf managed-flag {
           type boolean;
           description
             "The value that is placed in the 'Managed address
```

```
          configuration' flag field in the Router Advertisement.";
        }
        leaf other-config-flag {
          type boolean;
          description
            "The value that is placed in the 'Other configuration' flag
             field in the Router Advertisement.";
        }
        leaf link-mtu {
          type uint32;
          description
            "The value that is placed in MTU options sent by the
             router. A value of zero indicates that no MTU options are
             sent.";
        }
        leaf reachable-time {
          type uint32 {
            range "0..3600000";
          }
          units "milliseconds";
          description
            "The value that is placed in the Reachable Time field in
             the Router Advertisement messages sent by the router. A
             value of zero means unspecified (by this router).";
        }
        leaf retrans-timer {
          type uint32;
          units "milliseconds";
          description
            "The value that is placed in the Retrans Timer field in the
             Router Advertisement messages sent by the router. A value
             of zero means unspecified (by this router).";
        }
        leaf cur-hop-limit {
          type uint8;
          description
            "The value that is placed in the Cur Hop Limit field in the
             Router Advertisement messages sent by the router. A value
             of zero means unspecified (by this router).";
        }
        leaf default-lifetime {
          type uint16 {
            range "0..9000";
          }
          units "seconds";
          description
            "The value that is placed in the Router Lifetime field of
             Router Advertisements sent from the interface, in seconds.
```

```
           A value of zero indicates that the router is not to be
           used as a default router.";
        }
        container prefix-list {
          description
            "A list of prefixes that are placed in Prefix Information
             options in Router Advertisement messages sent from the
             interface.

             By default, these are all prefixes that the router
             advertises via routing protocols as being on-link for the
             interface from which the advertisement is sent.";
          list prefix {
            key "prefix-spec";
            description
              "Advertised prefix entry and its parameters.";
            leaf prefix-spec {
              type inet:ipv6-prefix;
              description
                "IPv6 address prefix.";
            }
            leaf valid-lifetime {
              type uint32;
              units "seconds";
              description
                "The value that is placed in the Valid Lifetime in the
                 Prefix Information option. The designated value of all
                 1's (0xffffffff) represents infinity.

                 An implementation SHOULD keep this value constant in
                 consecutive advertisements except when it is
                 explicitly changed in configuration.";
            }
            leaf on-link-flag {
              type boolean;
              description
                "The value that is placed in the on-link flag ('L-bit')
                 field in the Prefix Information option.";
            }
            leaf preferred-lifetime {
              type uint32;
              units "seconds";
              description
                "The value that is placed in the Preferred Lifetime in
                 the Prefix Information option, in seconds. The
                 designated value of all 1's (0xffffffff) represents
                 infinity.
```

```
                An implementation SHOULD keep this value constant in
                consecutive advertisements except when it is
                explicitly changed in configuration.";
            }
            leaf autonomous-flag {
              type boolean;
              description
                "The value that is placed in the Autonomous Flag field
                 in the Prefix Information option.";
            }
          }
        }
      }
    }

    /* Configuration data */

    augment "/if:interfaces/if:interface/ip:ipv6" {
      description
        "Augment interface configuration with parameters of IPv6 router
         advertisements.";
      container ipv6-router-advertisements {
        description
          "Configuration of IPv6 Router Advertisements.";
        leaf send-advertisements {
          type boolean;
          default "false";
          description
            "A flag indicating whether or not the router sends periodic
             Router Advertisements and responds to Router
             Solicitations.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvSendAdvertisements.";
        }
        leaf max-rtr-adv-interval {
          type uint16 {
            range "4..1800";
          }
          units "seconds";
          default "600";
          description
            "The maximum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             MaxRtrAdvInterval.";
        }
```

```
        leaf min-rtr-adv-interval {
          type uint16 {
            range "3..1350";
          }
          units "seconds";
          must ". <= 0.75 * ../max-rtr-adv-interval" {
            description
              "The value MUST NOT be greater than 75 % of
               'max-rtr-adv-interval'.";
          }
          description
            "The minimum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.

             The default value to be used operationally if this leaf is
             not configured is determined as follows:

             - if max-rtr-adv-interval >= 9 seconds, the default value
               is 0.33 * max-rtr-adv-interval;

             - otherwise it is 0.75 * max-rtr-adv-interval.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             MinRtrAdvInterval.";
        }
        leaf managed-flag {
          type boolean;
          default "false";
          description
            "The value to be placed in the 'Managed address
             configuration' flag field in the Router Advertisement.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvManagedFlag.";
        }
        leaf other-config-flag {
          type boolean;
          default "false";
          description
            "The value to be placed in the 'Other configuration' flag
             field in the Router Advertisement.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvOtherConfigFlag.";
        }
        leaf link-mtu {
          type uint32;
          default "0";
```

```
        description
          "The value to be placed in MTU options sent by the router.
           A value of zero indicates that no MTU options are sent.";
        reference
          "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
           AdvLinkMTU.";
      }
      leaf reachable-time {
        type uint32 {
          range "0..3600000";
        }
        units "milliseconds";
        default "0";
        description
          "The value to be placed in the Reachable Time field in the
           Router Advertisement messages sent by the router. A value
           of zero means unspecified (by this router).";
        reference
          "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
           AdvReachableTime.";
      }
      leaf retrans-timer {
        type uint32;
        units "milliseconds";
        default "0";
        description
          "The value to be placed in the Retrans Timer field in the
           Router Advertisement messages sent by the router. A value
           of zero means unspecified (by this router).";
        reference
          "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
           AdvRetransTimer.";
      }
      leaf cur-hop-limit {
        type uint8;
        description
          "The value to be placed in the Cur Hop Limit field in the
           Router Advertisement messages sent by the router. A value
           of zero means unspecified (by this router).

           If this parameter is not configured, the device SHOULD use
           the value specified in IANA Assigned Numbers that was in
           effect at the time of implementation.";
        reference
          "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
           AdvCurHopLimit.

           IANA: IP Parameters,
```

```
          http://www.iana.org/assignments/ip-parameters";
        }
        leaf default-lifetime {
          type uint16 {
            range "0..9000";
          }
          units "seconds";
          description
            "The value to be placed in the Router Lifetime field of
             Router Advertisements sent from the interface, in seconds.
             It MUST be either zero or between max-rtr-adv-interval and
             9000 seconds. A value of zero indicates that the router is
             not to be used as a default router. These limits may be
             overridden by specific documents that describe how IPv6
             operates over different link layers.

             If this parameter is not configured, the device SHOULD use
             a value of 3 * max-rtr-adv-interval.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvDefaultLifeTime.";
        }
        container prefix-list {
          description
            "Configuration of prefixes to be placed in Prefix
             Information options in Router Advertisement messages sent
             from the interface.

             Prefixes that are advertised by default but do not have
             their entries in the child 'prefix' list are advertised
             with the default values of all parameters.

             The link-local prefix SHOULD NOT be included in the list
             of advertised prefixes.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvPrefixList.";
          list prefix {
            key "prefix-spec";
            description
              "Configuration of an advertised prefix entry.";
            leaf prefix-spec {
              type inet:ipv6-prefix;
              description
                "IPv6 address prefix.";
            }
            choice control-adv-prefixes {
              default "advertise";
```

```
            description
              "The prefix either may be explicitly removed from the
               set of advertised prefixes, or parameters with which
               it is advertised may be specified (default case).";
            leaf no-advertise {
              type empty;
              description
                "The prefix will not be advertised.

                 This can be used for removing the prefix from the
                 default set of advertised prefixes.";
            }
            case advertise {
              leaf valid-lifetime {
                type uint32;
                units "seconds";
                default "2592000";
                description
                  "The value to be placed in the Valid Lifetime in
                   the Prefix Information option. The designated
                   value of all 1's (0xffffffff) represents
                   infinity.";
                reference
                  "RFC 4861: Neighbor Discovery for IP version 6
                   (IPv6) - AdvValidLifetime.";
              }
              leaf on-link-flag {
                type boolean;
                default "true";
                description
                  "The value to be placed in the on-link flag
                   ('L-bit') field in the Prefix Information
                   option.";
                reference
                  "RFC 4861: Neighbor Discovery for IP version 6
                   (IPv6) - AdvOnLinkFlag.";
              }
              leaf preferred-lifetime {
                type uint32;
                units "seconds";
                must ". <= ../valid-lifetime" {
                  description
                    "This value MUST NOT be greater than
                     valid-lifetime.";
                }
                default "604800";
                description
                  "The value to be placed in the Preferred Lifetime
```

```
                     in the Prefix Information option. The designated
                     value of all 1's (0xffffffff) represents
                     infinity.";
                   reference
                     "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvPreferredLifetime.";
                 }
                 leaf autonomous-flag {
                   type boolean;
                   default "true";
                   description
                     "The value to be placed in the Autonomous Flag
                     field in the Prefix Information option.";
                   reference
                     "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvAutonomousFlag.";
                 }
               }
             }
           }
         }
       }
     }
   }
```

<CODE ENDS>

## 10. IANA Considerations

RFC Ed.: In this section, replace all occurrences of 'XXXX' with the actual RFC number (and remove this note).

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

---

URI: urn:ietf:params:xml:ns:yang:ietf-routing

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

---

```
   ---------------------------------------------------------------------
   URI: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing

   Registrant Contact: The IESG.

   XML: N/A, the requested URI is an XML namespace.
   ---------------------------------------------------------------------

   ---------------------------------------------------------------------
   URI: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing

   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-routing
   namespace:    urn:ietf:params:xml:ns:yang:ietf-routing
   prefix:       rt
   reference:    RFC XXXX
   ---------------------------------------------------------------------

   ---------------------------------------------------------------------
   name:         ietf-ipv4-unicast-routing
   namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
   prefix:       v4ur
   reference:    RFC XXXX
   ---------------------------------------------------------------------

   ---------------------------------------------------------------------
   name:         ietf-ipv6-unicast-routing
   namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
   prefix:       v6ur
   reference:    RFC XXXX
   ---------------------------------------------------------------------
```

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

```
   ---------------------------------------------------------------------
   name:         ietf-ipv6-router-advertisements
   parent:       ietf-ipv6-unicast-routing
   reference:    RFC XXXX
   ---------------------------------------------------------------------
```

## 11. Security Considerations

Configuration and state data conforming to the core routing data model (defined in this document) are designed to be accessed via a management protocol with secure transport layer, such as NETCONF [RFC6241]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

A number of configuration data nodes defined in the YANG modules belonging to the core routing data model are writable/creatable/deletable (i.e., "config true" in YANG terms, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations to these data nodes, such as "edit-config" in NETCONF, can have negative effects on the network if the protocol operations are not properly protected.

The vulnerable "config true" parameters and subtrees are the following:

/routing/control-plane-protocols/control-plane-protocol: This list specifies the control plane protocols configured on a device.

/routing/ribs/rib: This list specifies the RIBs configured for the device.

Unauthorised access to any of these lists can adversely affect the routing subsystem of both the local device and the network. This may lead to network malfunctions, delivery of packets to inappropriate destinations and other problems.

## 12. Acknowledgments

The authors wish to thank Nitin Bahadur, Martin Bjorklund, Dean Bogdanovic, Jeff Haas, Joel Halpern, Wes Hardaker, Sriganesh Kini, David Lamparter, Andrew McGregor, Jan Medved, Xiang Li, Stephane Litkowski, Thomas Morin, Tom Petch, Yingzhen Qu, Bruno Rijsman, Juergen Schoenwaelder, Phil Shafer, Dave Thaler, Yi Yang, Derek Man-Kit Yeung and Jeffrey Zhang for their helpful comments and suggestions.

## 13. References

### 13.1. Normative References

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

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

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <http://www.rfc-editor.org/info/rfc4861>.

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

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

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

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

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

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

### 13.2. Informative References

[RFC6087] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", RFC 6087, DOI 10.17487/RFC6087, January 2011, <http://www.rfc-editor.org/info/rfc6087>.

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

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

## Appendix A. The Complete Data Trees

This appendix presents the complete configuration and state data trees of the core routing data model. See Section 2.2 for an explanation of the symbols used. Data type of every leaf node is shown near the right end of the corresponding line.

### A.1. Configuration Data

```
   +--rw routing
      +--rw router-id?                yang:dotted-quad
      +--rw control-plane-protocols
      |  +--rw control-plane-protocol* [type name]
      |     +--rw type             identityref
      |     +--rw name             string
      |     +--rw description?     string
      |     +--rw static-routes
      |        +--rw v6ur:ipv6
      |        |  +--rw v6ur:route* [destination-prefix]
      |        |     +--rw v6ur:destination-prefix    inet:ipv6-prefix
      |        |     +--rw v6ur:description?          string
      |        |     +--rw v6ur:next-hop
      |        |        +--rw (v6ur:next-hop-options)
      |        |           +--:(v6ur:simple-next-hop)
      |        |           |  +--rw v6ur:outgoing-interface?
      |        |           |  +--rw v6ur:next-hop-address?
      |        |           +--:(v6ur:special-next-hop)
      |        |           |  +--rw v6ur:special-next-hop?   enumeration
      |        |           +--:(v6ur:next-hop-list)
      |        |              +--rw v6ur:next-hop-list
      |        |                 +--rw v6ur:next-hop* [index]
      |        |                    +--rw v6ur:index              string
      |        |                    +--rw v6ur:outgoing-interface?
      |        |                    +--rw v6ur:next-hop-address?
      |        +--rw v4ur:ipv4
      |           +--rw v4ur:route* [destination-prefix]
      |              +--rw v4ur:destination-prefix    inet:ipv4-prefix
      |              +--rw v4ur:description?          string
      |              +--rw v4ur:next-hop
      |                 +--rw (v4ur:next-hop-options)
      |                    +--:(v4ur:simple-next-hop)
      |                    |  +--rw v4ur:outgoing-interface?
      |                    |  +--rw v4ur:next-hop-address?
      |                    +--:(v4ur:special-next-hop)
      |                    |  +--rw v4ur:special-next-hop?   enumeration
      |                    +--:(v4ur:next-hop-list)
      |                       +--rw v4ur:next-hop-list
      |                          +--rw v4ur:next-hop* [index]
      |                             +--rw v4ur:index              string
      |                             +--rw v4ur:outgoing-interface?
      |                             +--rw v4ur:next-hop-address?
      |
      +--rw ribs
         +--rw rib* [name]
            +--rw name              string
            +--rw address-family?   identityref
            +--rw description?      string
```

A.2. State Data

```
+--ro routing-state
|  +--ro router-id?                yang:dotted-quad
|  +--ro interfaces
|  |  +--ro interface*   if:interface-state-ref
|  +--ro control-plane-protocols
|  |  +--ro control-plane-protocol* [type name]
|  |     +--ro type    identityref
|  |     +--ro name    string
|  +--ro ribs
|     +--ro rib* [name]
|        +--ro name              string
|        +--ro address-family    identityref
|        +--ro default-rib?      boolean {multiple-ribs}?
|        +--ro routes
|        |  +--ro route*
|        |     +--ro route-preference?          route-preference
|        |     +--ro next-hop
|        |     |  +--ro (next-hop-options)
|        |     |     +--:(simple-next-hop)
|        |     |     |  +--ro outgoing-interface?
|        |     |     |  +--ro v6ur:next-hop-address?
|        |     |     |  +--ro v4ur:next-hop-address?
|        |     |     +--:(special-next-hop)
|        |     |     |  +--ro special-next-hop?        enumeration
|        |     |     +--:(next-hop-list)
|        |     |        +--ro next-hop-list
|        |     |           +--ro next-hop*
|        |     |              +--ro outgoing-interface?
|        |     |              +--ro v6ur:address?
|        |     |              +--ro v4ur:address?
|        |     +--ro source-protocol            identityref
|        |     +--ro active?                    empty
|        |     +--ro last-updated?              yang:date-and-time
|        |     +--ro v6ur:destination-prefix?   inet:ipv6-prefix
|        |     +--ro v4ur:destination-prefix?   inet:ipv4-prefix
|        +---x active-route
|           +---w input
|           |  +---w v6ur:destination-address?   inet:ipv6-address
|           |  +---w v4ur:destination-address?   inet:ipv4-address
|           +--ro output
|              +--ro route
|                 +--ro next-hop
|                 |  +--ro (next-hop-options)
|                 |     +--:(simple-next-hop)
|                 |     |  +--ro outgoing-interface?
|                 |     |  +--ro v6ur:next-hop-address?
```

```
|                |     |  +--ro v4ur:next-hop-address?
|                |     +--:(special-next-hop)
|                |     |  +--ro special-next-hop?     enumeration
|                |     +--:(next-hop-list)
|                |        +--ro next-hop-list
|                |           +--ro next-hop*
|                |              +--ro outgoing-interface?
|                |              +--ro v6ur:next-hop-address?
|                |              +--ro v4ur:next-hop-address?
|                +--ro source-protocol            identityref
|                +--ro active?                    empty
|                +--ro last-updated?           yang:date-and-time
|                +--ro v6ur:destination-prefix?  inet:ipv6-prefix
|                +--ro v4ur:destination-prefix?  inet:ipv4-prefix
```

## Appendix B. Minimum Implementation

Some parts and options of the core routing model, such as user-defined RIBs, are intended only for advanced routers. This appendix gives basic non-normative guidelines for implementing a bare minimum of available functions. Such an implementation may be used for hosts or very simple routers.

A minimum implementation does not support the feature "multiple-ribs". This means that a single system-controlled RIB is available for each supported address family - IPv4, IPv6 or both. These RIBs are also the default RIBs. No user-controlled RIBs are allowed.

In addition to the mandatory instance of the "direct" pseudo-protocol, a minimum implementation should support configuring instance(s) of the "static" pseudo-protocol.

For hosts that are never intended to act as routers, the ability to turn on sending IPv6 router advertisements (Section 5.4) should be removed.

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" control plane protocol instances.

## Appendix C. Example: Adding a New Control Plane Protocol

This appendix demonstrates how the core routing data model can be extended to support a new control plane protocol. The YANG module "example-rip" shown below is intended as an illustration rather than a real definition of a data model for the RIP routing protocol. For the sake of brevity, this module does not obey all the guidelines specified in [RFC6087]. See also Section 5.3.2.

```
   module example-rip {

     yang-version "1.1";

     namespace "http://example.com/rip";

     prefix "rip";

     import ietf-interfaces {
       prefix "if";
     }

     import ietf-routing {
       prefix "rt";
     }

     identity rip {
       base rt:routing-protocol;
       description
         "Identity for the RIP routing protocol.";
     }

     typedef rip-metric {
       type uint8 {
         range "0..16";
       }
     }

     grouping route-content {
       description
         "This grouping defines RIP-specific route attributes.";
       leaf metric {
         type rip-metric;
       }
       leaf tag {
         type uint16;
         default "0";
         description
           "This leaf may be used to carry additional info, e.g. AS
            number.";
       }
     }

     augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
       when "derived-from-or-self(rt:source-protocol, 'rip:rip')" {
         description
           "This augment is only valid for a routes whose source
            protocol is RIP.";
```

```
      }
      description
        "RIP-specific route attributes.";
      uses route-content;
    }

    augment "/rt:routing-state/rt:ribs/rt:rib/rt:active-route/"
          + "rt:output/rt:route" {
      description
        "RIP-specific route attributes in the output of 'active-route'
         RPC.";
      uses route-content;
    }

    augment "/rt:routing/rt:control-plane-protocols/"
          + "rt:control-plane-protocol" {
      when "derived-from-or-self(rt:type,'rip:rip')" {
        description
          "This augment is only valid for a routing protocol instance
           of type 'rip'.";
      }
      container rip {
        presence "RIP configuration";
        description
          "RIP instance configuration.";
        container interfaces {
          description
            "Per-interface RIP configuration.";
          list interface {
            key "name";
            description
              "RIP is enabled on interfaces that have an entry in this
               list, unless 'enabled' is set to 'false' for that
               entry.";
            leaf name {
              type if:interface-ref;
            }
            leaf enabled {
              type boolean;
              default "true";
            }
            leaf metric {
              type rip-metric;
              default "1";
            }
          }
        }
        leaf update-interval {
```

```
          type uint8 {
            range "10..60";
          }
          units "seconds";
          default "30";
          description
            "Time interval between periodic updates.";
        }
      }
    }
  }
```

## Appendix D. Data Tree Example

This section contains an example instance data tree in the JSON encoding [RFC7951], containing both configuration and state data. The data conforms to a data model that is defined by the following YANG library specification [RFC7895]:

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "c2e1f54169aa7f36e1a6e8d0865d441d3600f9c4",
    "module": [
      {
        "name": "ietf-routing",
        "revision": "2016-11-03",
        "feature": [
          "multiple-ribs",
          "router-id"
        ],
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv4-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv6-unicast-routing",
        "revision": "2016-11-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
        "conformance-type": "implement"
      },
      {
```

```
          "name": "ietf-interfaces",
          "revision": "2014-05-08",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-inet-types",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
          "revision": "2013-07-15",
          "conformance-type": "import"
        },
        {
          "name": "ietf-yang-types",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
          "revision": "2013-07-15",
          "conformance-type": "import"
        },
        {
          "name": "iana-if-type",
          "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
          "revision": "",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        }
      ]
    }
  }
```

A simple network set-up as shown in Figure 3 is assumed: router "A" uses static default routes with the "ISP" router as the next-hop. IPv6 router advertisements are configured only on the "eth1" interface and disabled on the upstream "eth0" interface.

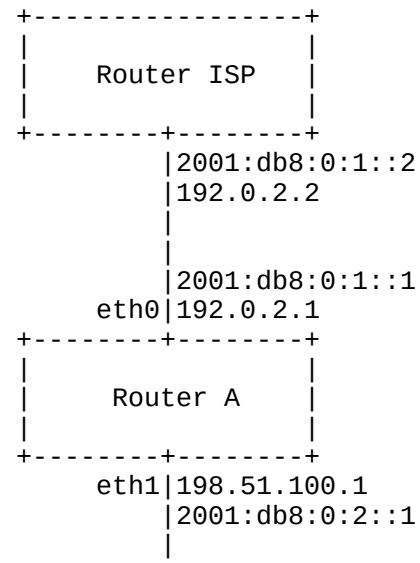



Figure 3: Example network configuration

The instance data tree could then be as follows:

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "description": "Uplink to ISP.",
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.1",
              "prefix-length": 24
            }
          ],
          "forwarding": true
        },
        "ietf-ip:ipv6": {
          "address": [
            {
              "ip": "2001:0db8:0:1::1",
              "prefix-length": 64
            }
          ],
          "forwarding": true,
```

```
            "autoconf": {
              "create-global-addresses": false
            }
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "description": "Interface to the internal network.",
          "ietf-ip:ipv4": {
            "address": [
              {
                "ip": "198.51.100.1",
                "prefix-length": 24
              }
            ],
            "forwarding": true
          },
          "ietf-ip:ipv6": {
            "address": [
              {
                "ip": "2001:0db8:0:2::1",
                "prefix-length": 64
              }
            ],
            "forwarding": true,
            "autoconf": {
              "create-global-addresses": false
            },
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": true
            }
          }
        }
      ]
    },
    "ietf-interfaces:interfaces-state": {
      "interface": [
        {
          "name": "eth0",
          "type": "iana-if-type:ethernetCsmacd",
          "phys-address": "00:0C:42:E5:B1:E9",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2015-10-24T17:11:27+02:00"
          },
          "ietf-ip:ipv4": {
            "forwarding": true,
```

```
            "mtu": 1500,
            "address": [
              {
                "ip": "192.0.2.1",
                "prefix-length": 24
              }
            ]
          },
          "ietf-ip:ipv6": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:1::1",
                "prefix-length": 64
              }
            ],
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": true,
              "prefix-list": {
                "prefix": [
                  {
                    "prefix-spec": "2001:db8:0:2::/64"
                  }
                ]
              }
            }
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "phys-address": "00:0C:42:E5:B1:EA",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2015-10-24T17:11:29+02:00"
          },
          "ietf-ip:ipv4": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "198.51.100.1",
                "prefix-length": 24
              }
            ]
          },
          "ietf-ip:ipv6": {
```

```
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:2::1",
                "prefix-length": 64
              }
            ],
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": true,
              "prefix-list": {
                "prefix": [
                  {
                    "prefix-spec": "2001:db8:0:2::/64"
                  }
                ]
              }
            }
          }
        }
      ]
    },
    "ietf-routing:routing": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:static",
            "name": "st0",
            "description":
              "Static routing is used for the internal network.",
            "static-routes": {
              "ietf-ipv4-unicast-routing:ipv4": {
                "route": [
                  {
                    "destination-prefix": "0.0.0.0/0",
                    "next-hop": {
                      "next-hop-address": "192.0.2.2"
                    }
                  }
                ]
              },
              "ietf-ipv6-unicast-routing:ipv6": {
                "route": [
                  {
                    "destination-prefix": "::/0",
                    "next-hop": {
                      "next-hop-address": "2001:db8:0:1::2"
```

```
                    }
                  }
                ]
              }
            }
          }
        ]
      }
    },
    "ietf-routing:routing-state": {
      "interfaces": {
        "interface": [
          "eth0",
          "eth1"
        ]
      },
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:static",
            "name": "st0"
          }
        ]
      },
      "ribs": {
        "rib": [
          {
            "name": "ipv4-master",
            "address-family":
              "ietf-ipv4-unicast-routing:ipv4-unicast",
            "default-rib": true,
            "routes": {
              "route": [
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "192.0.2.1/24",
                  "next-hop": {
                    "outgoing-interface": "eth0"
                  },
                  "route-preference": 0,
                  "source-protocol": "ietf-routing:direct",
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "198.51.100.0/24",
                  "next-hop": {
                    "outgoing-interface": "eth1"
```

```
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "0.0.0.0/0",
                  "source-protocol": "ietf-routing:static",
                  "route-preference": 5,
                  "next-hop": {
                    "ietf-ipv4-unicast-routing:next-hop-address":
                      "192.0.2.2"
                  },
                  "last-updated": "2015-10-24T18:02:45+02:00"
                }
              ]
            }
          },
          {
            "name": "ipv6-master",
            "address-family":
              "ietf-ipv6-unicast-routing:ipv6-unicast",
            "default-rib": true,
            "routes": {
              "route": [
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
                    "2001:db8:0:1::/64",
                  "next-hop": {
                    "outgoing-interface": "eth0"
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
                    "2001:db8:0:2::/64",
                  "next-hop": {
                    "outgoing-interface": "eth1"
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
```

```
                    "::/0",
                  "next-hop": {
                    "ietf-ipv6-unicast-routing:next-hop-address":
                      "2001:db8:0:1::2"
                  },
                  "source-protocol": "ietf-routing:static",
                  "route-preference": 5,
                  "last-updated": "2015-10-24T18:02:45+02:00"
                }
              ]
            }
          }
        ]
      }
    }
  }
```

## Appendix E. Change Log

RFC Editor: Remove this section upon publication as an RFC.

### E.1. Changes Between Versions -24 and -25

- Minor edits based on IETF Last Call reviews.

### E.2. Changes Between Versions -23 and -24

- Fix paths in "when" expressions due to errata 4749 of RFC 7950.

### E.3. Changes Between Versions -22 and -23

- Removed "route-tag" feature.
- Removed next-hop classifiers.
- Fixed invalid when expressions in augments.
- In simple-next-hop, an address, outgoing interface or both can be specified.
- RPC "fib-route" changed into RIB action "active-route".
- The requirement that direct routes be always placed in default RIBs.

### E.4. Changes Between Versions -21 and -22

- Added "next-hop-list" as a new case of the "next-hop-options" choice.
- Renamed "routing protocol" to "control plane protocol" in both the YANG modules and I-D text.

### E.5. Changes Between Versions -20 and -21

- Routing instances were removed.
- IPv6 RA parameters were moved to the "ietf-ipv6-router-advertisements".

### E.6. Changes Between Versions -19 and -20

- Assignment of L3 interfaces to routing instances is now part of interface configuration.
- Next-hop options in configuration were aligned with state data.
- It is recommended to enclose protocol-specific configuration in a presence container.

### E.7. Changes Between Versions -18 and -19

- The leaf "route-preference" was removed from the "routing-protocol" container in both "routing" and "routing-state".
- The "vrf-routing-instance" identity was added in support of a common routing-instance type in addition to the "default-routing-instance".
- Removed "enabled" switch from "routing-protocol".

### E.8. Changes Between Versions -17 and -18

- The container "ribs" was moved under "routing-instance" (in both "routing" and "routing-state").
- Typedefs "rib-ref" and "rib-state-ref" were removed.
- Removed "recipient-ribs" (both state and configuration).
- Removed "connected-ribs" from "routing-protocol" (both state and configuration).

- Configuration and state data for IPv6 RA were moved under "if:interface" and "if:interface-state".

- Assignment of interfaces to routing instances now use leaf-list rather than list (both config and state). The opposite reference from "if:interface" to "rt:routing-instance" was changed to a single leaf (an interface cannot belong to multiple routing instances).

- Specification of a default RIB is now a simple flag under "rib" (both config and state).

- Default RIBs are marked by a flag in state data.

### E.9. Changes Between Versions -16 and -17

- Added Acee as a co-author.

- Removed all traces of route filters.

- Removed numeric IDs of list entries in state data.

- Removed all next-hop cases except "simple-next-hop" and "special-next-hop".

- Removed feature "multipath-routes".

- Augmented "ietf-interfaces" module with a leaf-list of leafrefs pointing form state data of an interface entry to the routing instance(s) to which the interface is assigned.

### E.10. Changes Between Versions -15 and -16

- Added ’type’ as the second key component of ’routing-protocol’, both in configuration and state data.

- The restriction of no more than one connected RIB per address family was removed.

- Removed the ’id’ key of routes in RIBs. This list has no keys anymore.

- Remove the ’id’ key from static routes and make ’destination-prefix’ the only key.

- Added ’route-preference’ as a new attribute of routes in RIB.

- Added ’active’ as a new attribute of routes in RIBs.

- Renamed RPC operation 'active-route' to 'fib-route'.
- Added 'route-preference' as a new parameter of routing protocol instances, both in configuration and state data.
- Renamed identity 'rt:standard-routing-instance' to 'rt:default-routing-instance'.
- Added next-hop lists to state data.
- Added two cases for specifying next-hops indirectly - via a new RIB or a recursive list of next-hops.
- Reorganized next-hop in static routes.
- Removed all 'if-feature' statements from state data.

### E.11. Changes Between Versions -14 and -15

- Removed all defaults from state data.
- Removed default from 'cur-hop-limit' in config.

### E.12. Changes Between Versions -13 and -14

- Removed dependency of 'connected-ribs' on the 'multiple-ribs' feature.
- Removed default value of 'cur-hop-limit' in state data.
- Moved parts of descriptions and all references on IPv6 RA parameters from state data to configuration.
- Added reference to RFC 6536 in the Security section.

### E.13. Changes Between Versions -12 and -13

- Wrote appendix about minimum implementation.
- Remove "when" statement for IPv6 router interface state data - it was dependent on a config value that may not be present.
- Extra container for the next-hop list.
- Names rather than numeric ids are used for referring to list entries in state data.

- Numeric ids are always declared as mandatory and unique. Their description states that they are ephemeral.
- Descriptions of "name" keys in state data lists are required to be persistent.
- 
- Removed "if-feature multiple-ribs;" from connected-ribs.
- "rib-name" instead of "name" is used as the name of leafref nodes.
- "next-hop" instead of "nexthop" or "gateway" used throughout, both in node names and text.

### E.14. Changes Between Versions -11 and -12

- Removed feature "advanced-router" and introduced two features instead: "multiple-ribs" and "multipath-routes".
- Unified the keys of config and state versions of "routing-instance" and "rib" lists.
- Numerical identifiers of state list entries are not keys anymore, but they are constrained using the "unique" statement.
- Updated acknowledgements.

### E.15. Changes Between Versions -10 and -11

- Migrated address families from IANA enumerations to identities.
- Terminology and node names aligned with the I2RS RIB model: router -> routing instance, routing table -> RIB.
- Introduced uint64 keys for state lists: routing-instance, rib, route, nexthop.
- Described the relationship between system-controlled and user-controlled list entries.
- Feature "user-defined-routing-tables" changed into "advanced-router".
- Made nexthop into a choice in order to allow for nexthop-list (I2RS requirement).

- Added nexthop-list with entries having priorities (backup) and weights (load balancing).

- Updated bibliography references.

### E.16. Changes Between Versions -09 and -10

- Added subtree for state data ("/routing-state").

- Terms "system-controlled entry" and "user-controlled entry" defined and used.

- New feature "user-defined-routing-tables". Nodes that are useful only with user-defined routing tables are now conditional.

- Added grouping "router-id".

- In routing tables, "source-protocol" attribute of routes now reports only protocol type, and its datatype is "identityref".

- Renamed "main-routing-table" to "default-routing-table".

### E.17. Changes Between Versions -08 and -09

- Fixed "must" expression for "connected-routing-table".

- Simplified "must" expression for "main-routing-table".

- Moved per-interface configuration of a new routing protocol under 'routing-protocol'. This also affects the 'example-rip' module.

### E.18. Changes Between Versions -07 and -08

- Changed reference from RFC6021 to RFC6021bis.

### E.19. Changes Between Versions -06 and -07

- The contents of <get-reply> in Appendix D was updated: "eth[01]" is used as the value of "location", and "forwarding" is on for both interfaces and both IPv4 and IPv6.

- The "must" expression for "main-routing-table" was modified to avoid redundant error messages reporting address family mismatch when "name" points to a non-existent routing table.

- The default behavior for IPv6 RA prefix advertisements was clarified.

- Changed type of "rt:router-id" to "ip:dotted-quad".
- Type of "rt:router-id" changed to "yang:dotted-quad".
- Fixed missing prefixes in XPath expressions.

### E.20. Changes Between Versions -05 and -06

- Document title changed: "Configuration" was replaced by "Management".
- New typedefs "routing-table-ref" and "route-filter-ref".
- Double slashes "//" were removed from XPath expressions and replaced with the single "/".
- Removed uniqueness requirement for "router-id".
- Complete data tree is now in Appendix A.
- Changed type of "source-protocol" from "leafref" to "string".
- Clarified the relationship between routing protocol instances and connected routing tables.
- Added a must constraint saying that a routing table connected to the direct pseudo-protocol must not be a main routing table.

### E.21. Changes Between Versions -04 and -05

- Routing tables are now global, i.e., "routing-tables" is a child of "routing" rather than "router".
- "must" statement for "static-routes" changed to "when".
- Added "main-routing-tables" containing references to main routing tables for each address family.
- Removed the defaults for "address-family" and "safi" and made them mandatory.
- Removed the default for route-filter/type and made this leaf mandatory.
- If there is no active route for a given destination, the "active-route" RPC returns no output.
- Added "enabled" switch under "routing-protocol".

- Added "router-type" identity and "type" leaf under "router".
- Route attribute "age" changed to "last-updated", its type is "yang:date-and-time".
- The "direct" pseudo-protocol is always connected to main routing tables.
- Entries in the list of connected routing tables renamed from "routing-table" to "connected-routing-table".
- Added "must" constraint saying that a routing table must not be its own recipient.

### E.22. Changes Between Versions -03 and -04

- Changed "error-tag" for both RPC operations from "missing element" to "data-missing".
- Removed the decrementing behavior for advertised IPv6 prefix parameters "valid-lifetime" and "preferred-lifetime".
- Changed the key of the static route lists from "seqno" to "id" because the routes needn’t be sorted.
- Added ’must’ constraint saying that "preferred-lifetime" must not be greater than "valid-lifetime".

### E.23. Changes Between Versions -02 and -03

- Module "iana-afn-safi" moved to I-D "iana-if-type".
- Removed forwarding table.
- RPC "get-route" changed to "active-route". Its output is a list of routes (for multi-path routing).
- New RPC "route-count".
- For both RPCs, specification of negative responses was added.
- Relaxed separation of router instances.
- Assignment of interfaces to router instances needn’t be disjoint.
- Route filters are now global.
- Added "allow-all-route-filter" for symmetry.

- Added Section 6 about interactions with "ietf-interfaces" and "ietf-ip".
- Added "router-id" leaf.
- Specified the names for IPv4/IPv6 unicast main routing tables.
- Route parameter "last-modified" changed to "age".
- Added container "recipient-routing-tables".

### E.24. Changes Between Versions -01 and -02

- Added module "ietf-ipv6-unicast-routing".
- The example in Appendix D now uses IP addresses from blocks reserved for documentation.
- Direct routes appear by default in the forwarding table.
- Network layer interfaces must be assigned to a router instance. Additional interface configuration may be present.
- The "when" statement is only used with "augment", "must" is used elsewhere.
- Additional "must" statements were added.
- The "route-content" grouping for IPv4 and IPv6 unicast now includes the material from the "ietf-routing" version via "uses rt:route-content".
- Explanation of symbols in the tree representation of data model hierarchy.

### E.25. Changes Between Versions -00 and -01

- AFN/SAFI-independent stuff was moved to the "ietf-routing" module.
- Typedefs for AFN and SAFI were placed in a separate "iana-afn-safi" module.
- Names of some data nodes were changed, in particular "routing-process" is now "router".
- The restriction of a single AFN/SAFI per router was lifted.
- RPC operation "delete-route" was removed.

- Illegal XPath references from "get-route" to the datastore were fixed.
- Section "Security Considerations" was written.

## Authors' Addresses

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Acee Lindem
Cisco Systems

Email: acee@cisco.com

Routing Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 16, 2018

S. Litkowski
B. Decraene
Orange
C. Filsfils
Cisco Systems
P. Francois
Individual

November 12, 2017

# Micro-loop prevention by introducing a local convergence delay
## draft-ietf-rtgwg-uloop-delay-09

## Abstract

This document describes a mechanism for link-state routing protocols to prevent local transient forwarding loops in case of link failure. This mechanism proposes a two-step convergence by introducing a delay between the convergence of the node adjacent to the topology change and the network wide convergence.

As this mechanism delays the IGP convergence it may only be used for planned maintenance or when fast reroute protects the traffic between the link failure time and the IGP convergence.

The proposed mechanism is limited to the link down event in order to keep the mechanism simple.

Simulations using real network topologies have been performed and show that local loops are a significant portion (>50%) of the total forwarding loops.

## Requirements Language

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

## 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/.

## Copyright Notice



## Table of Contents

## 1. Acronyms

FIB: Forwarding Information Base

FRR: Fast ReRoute

IGP: Interior Gateway Protocol

LFA: Loop Free Alternate

LSA: Link State Advertisement

LSP: Link State Packet

MRT: Maximum Redundant Trees

OFIB: Ordered FIB

PLSN: Path Locking via Safe Neighbor

RIB: Routing Information Base

RLFA: Remote Loop Free Alternate

SPF: Shortest Path First

TTL: Time To Live

## 2. Introduction

Micro-forwarding loops and some potential solutions are well described in [RFC5715]. This document describes a simple targeted mechanism that prevents micro-loops that are local to the failure. Based on network analysis, local failures make up a significant portion of the micro-forwarding loops. A simple and easily deployable solution for these local micro-loops is critical because these local loops cause some traffic loss after a fast-reroute alternate has been used (see Section 3.1).

Consider the case in Figure 1 where S does not have an LFA (Loop Free Alternate) to protect its traffic to D when the S-D link fails. That means that all non-D neighbors of S on the topology will send to S any traffic destined to D; if a neighbor did not, then that neighbor would be loop-free. Regardless of the advanced fast-reroute (FRR) technique used, when S converges to the new topology, it will send its traffic to a neighbor that was not loop-free and thus cause a local micro-loop. The deployment of advanced fast-reroute techniques motivates this simple router-local mechanism to solve this targeted problem. This solution can work with the various techniques described in [RFC5715].

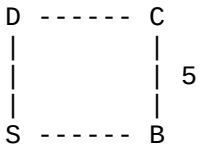



Figure 1

In the Figure 1, all links have a metric of 1 except B-C which has a metric of 5. When S-D fails, a transient forwarding loop may appear between S and B if S updates its forwarding entry to D before B does.

## 3. Transient forwarding loops side effects

Even if they are very limited in duration, transient forwarding loops may cause significant network damage.

### 3.1. Fast reroute inefficiency

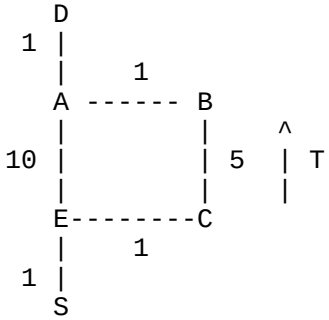



Figure 2 - RSVP-TE FRR case

In the Figure 2, we consider an IP/LDP routed network. An RSVP-TE tunnel T, provisioned on C and terminating on B, is used to protect

the traffic against C-B link failure (the IGP shortcut feature, defined in [RFC3906], is activated on C ). The primary path of T is C->B and FRR is activated on T providing an FRR bypass or detour using path C->E->A->B. On router C, the next hop to D is the tunnel T thanks to the IGP shortcut. When C-B link fails:

1. C detects the failure, and updates the tunnel path using a preprogrammed FRR path. The traffic path from S to D becomes: S->E->C->E->A->B->A->D.

2. In parallel, on router C, both the IGP convergence and the TE tunnel convergence (tunnel path recomputation) are occurring:

   * The Tunnel T path is recomputed and now uses C->E->A->B.

   * The IGP path to D is recomputed and now uses C->E->A->D.

3. On C, the tail-end of the TE tunnel (router B) is no longer on the shortest-path tree (SPT) to D, so C does not continue to encapsulate the traffic to D using the tunnel T and updates its forwarding entry to D using the nexthop E.

If C updates its forwarding entry to D before router E, there would be a transient forwarding loop between C and E until E has converged.

The table 1 below describes a theoretical sequence of events happening when the B-C link fails. This theoretical sequence of events should only be read as an example.

| Network condition | Time | Router C events | Router E events |
|---|---|---|---|
| S->D Traffic OK | | | |
| S->D Traffic lost | t0 | Link B-C fails | Link B-C fails |
| | t0+20msec | C detects the failure | |
| S->D Traffic OK | t0+40msec | C activates FRR | |

| | | | | |
|---|---|---|---|---|
| | t0+50msec | C updates its local LSP/LSA | | |
| | t0+60msec | C schedules SPF (100ms) | | |
| | t0+70msec | C floods its local updated LSP/LSA | | |
| | t0+87msec | | E receives LSP/LSA from C and schedules SPF (100ms) | |
| | t0+117msec | | E floods LSP/LSA from C | |
| | t0+160msec | C computes SPF | | |
| | t0+165msec | C starts updating its RIB/FIB | | |
| | t0+193msec | | E computes SPF | |
| | t0+199msec | | E starts updating its RIB/FIB | |
| S->D Traffic lost | t0+255msec | C updates its RIB/FIB for D | | |
| | t0+340msec | C convergence ends | | |
| S->D Traffic OK | t0+443msec | | E updates its RIB/FIB for D | |
| | t0+470msec | | E convergence ends | |

Table 1 - Route computation event time scale

The issue described here is completely independent of the fast-reroute mechanism involved (TE FRR, LFA/rLFA, MRT ...) when the primary path uses hop-by-hop routing. The protection enabled by fast-reroute is working perfectly, but ensures a protection, by

definition, only until the PLR has converged (as soon as the PLR has converged, it replaces its FRR path by a new primary path). When implementing FRR, a service provider wants to guarantee a very limited loss of connectivity time. The previous example shows that the benefit of FRR may be completely lost due to a transient forwarding loop appearing when PLR has converged. Delaying FIB updates after the IGP convergence may allow to keep the fast-reroute path until the neighbors have converged and preserves the customer traffic.

## 3.2. Network congestion

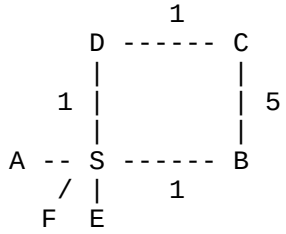



Figure 3

In the figure above, as presented in Section 2, when the link S-D fails, a transient forwarding loop may appear between S and B for destination D. The traffic on the S-B link will constantly increase due to the looping traffic to D. Depending on the TTL of the packets, the traffic rate destined to D, and the bandwidth of the link, the S-B link may become congested in a few hundreds of milliseconds and will stay congested until the loop is eliminated.

The congestion introduced by transient forwarding loops is problematic as it can affect traffic that is not directly affected by the failing network component. In the example, the congestion of the S-B link will impact some customer traffic that is not directly affected by the failure: e.g. A to B, F to B, E to B. Class of service may mitigate the congestion for some traffic. However, some traffic not directly affected by the failure will still be dropped as a router is not able to distinguish the looping traffic from the normally forwarded traffic.

# 4. Overview of the solution

This document defines a two-step convergence initiated by the router detecting a failure and advertising the topological changes in the IGP. This introduces a delay between network-wide convergence and the convergence of the local router.

The proposed solution is limited to local link down events in order to keep the solution simple.

This ordered convergence is similar to the ordered FIB proposed defined in [RFC6976], but it is limited to only a "one hop" distance. As a consequence, it is more simple and becomes a local-only feature that does not require interoperability. This benefit comes with the limitation of eliminating transient forwarding loops involving the local router only. The proposed mechanism also reuses some concepts described in [I-D.ietf-rtgwg-microloop-analysis].

## 5. Specification

### 5.1. Definitions

This document will refer to the following existing IGP timers. These timers may be standardized or implemented as a vendor specific local feature.

- LSP_GEN_TIMER: The delay between two consecutives local LSP/LSA generation. From an operational point of view, this delay is usually tuned to batch multiple local events in one single local LSP/LSA update. In IS-IS, this timer is defined as minimumLSPGenerationInterval in [ISO10589]. In OSPF version 2, this timer is defined as MinLSInterval in [RFC2328]. It is often associated with a vendor specific damping mechanism to slow down reactions by incrementing the timer when multiple consecutive events are detected.

- SPF_DELAY: The delay between the first IGP event triggering a new routing table computation and the start of that routing table computation. It is often associated with a damping mechanism to slow down reactions by incrementing the timer when the IGP becomes unstable. As an example, [I-D.ietf-rtgwg-backoff-algo] defines a standard SPF (Shortest Path First) delay algorithm.

This document introduces the following new timer:

- ULOOP_DELAY_DOWN_TIMER: used to slow down the local node convergence in case of link down events.

### 5.2. Regular IGP reaction

Upon a change of the status of an adjacency/link, the regular IGP convergence behavior of the router advertising the event involves the following main steps:

1. IGP is notified of the Up/Down event.

2. The IGP processes the notification and postpones the reaction for LSP_GEN_TIMER msec.

3. Upon LSP_GEN_TIMER expiration, the IGP updates its LSP/LSA and floods it.

4. The SPF computation is scheduled in SPF_DELAY msec.

5. Upon SPF_DELAY timer expiration, the SPF is computed, then the RIB and FIB are updated.

### 5.3. Local events

The mechanism described in this document assumes that there has been a single link failure as seen by the IGP area/level. If this assumption is violated (e.g. multiple links or nodes failed), then regular IP convergence must be applied (as described in Section 5.2).

To determine if the mechanism can be applicable or not, an implementation SHOULD implement logic to correlate the protocol messages (LSP/LSA) received during the SPF scheduling period in order to determine the topology changes that occured. This is necessary as multiple protocol messages may describe the same topology change and a single protocol message may describe multiple topology changes. As a consequence, determining a particular topology change MUST be independent of the order of reception of those protocol messages. How the logic works is left to the implementation.

Using this logic, if an implementation determines that the associated topology change is a single local link failure, then the router MAY use the mechanism described in this document, otherwise the regular IP convergence MUST be used.

Example:

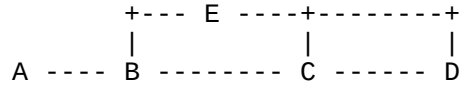



Figure 4

Let router B be the computing router when the link B-C fails. B updates its local LSP/LSA describing the link B->C as down, C does the same, and both start flooding their updated LSP/LSAs. During the SPF_DELAY period, B and C learn all the LSPs/LSAs to consider. B sees that C is flooding an advertisement that indicates that a link is down, and B is the other end of that link. B determines that B and C are describing the same single event. Since B receives no

other changes, B can determine that this is a local link failure and may decide to activate the mechanism described in this document.

### 5.4. Local delay for link down

Upon an adjacency/link down event, this document introduces a change in step 5 (Section 5.2) in order to delay the local convergence compared to the network wide convergence. The new step 5 is described below:

> 5. Upon SPF_DELAY timer expiration, the SPF is computed. If the condition of a single local link-down event has been met, then an update of the RIB and the FIB MUST be delayed for ULOOP_DELAY_DOWN_TIMER msecs. Otherwise, the RIB and FIB SHOULD be updated immediately.

If a new convergence occurs while ULOOP_DELAY_DOWN_TIMER is running, ULOOP_DELAY_DOWN_TIMER is stopped and the RIB/FIB SHOULD be updated as part of the new convergence event.

As a result of this addition, routers local to the failure will converge slower than remote routers. Hence it SHOULD only be done for a non-urgent convergence, such as for administrative de-activation (maintenance) or when the traffic is protected by fast-reroute.

## 6. Applicability

As previously stated, this mechanism only avoids the forwarding loops on the links between the node local to the failure and its neighbors. Forwarding loops may still occur on other links.

### 6.1. Applicable case: local loops

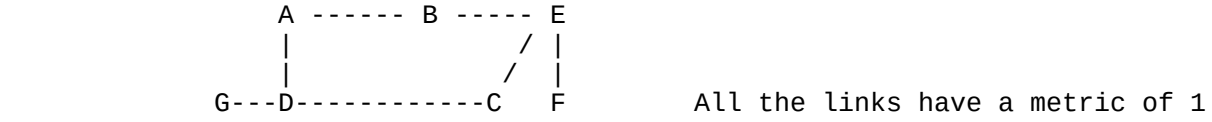



Figure 5

Let us consider the traffic from G to F. The primary path is G->D->C->E->F. When link C-E fails, if C updates its forwarding entry for F before D, a transient loop occurs. This is sub-optimal as C has FRR enabled and it breaks the FRR forwarding while all upstream routers are still forwarding the traffic to itself.

By implementing the mechanism defined in this document on C, when the C-E link fails, C delays the update of its forwarding entry to F, in order to allow some time for D to converge. FRR on C keeps protecting the traffic during this period. When the timer expires on C, its forwarding entry to F is updated. There is no transient forwarding loop on the link C-D.

### 6.2. Non applicable case: remote loops

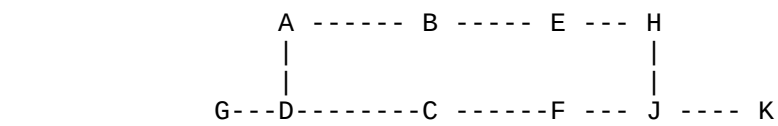



Figure 6

Let us consider the traffic from G to K. The primary path is G->D->C->F->J->K. When the C-F link fails, if C updates its forwarding entry to K before D, a transient loop occurs between C and D.

By implementing the mechanism defined in this document on C, when the link C-F fails, C delays the update of its forwarding entry to K, allowing time for D to converge. When the timer expires on C, its forwarding entry to F is updated. There is no transient forwarding loop between C and D. However, a transient forwarding loop may still occur between D and A. In this scenario, this mechanism is not enough to address all the possible forwarding loops. However, it does not create additional traffic loss. Besides, in some cases -such as when the nodes update their FIB in the following order C, A, D, for example because the router A is quicker than D to converge- the mechanism may still avoid the forwarding loop that would have otherwise occurred.

## 7. Simulations

Simulations have been run on multiple service provider topologies.

| Topology | Gain |
|---|---|
| T1 | 71% |
| T2 | 81% |
| T3 | 62% |
| T4 | 50% |
| T5 | 70% |
| T6 | 70% |
| T7 | 59% |
| T8 | 77% |

Table 2 - Number of Repair/Dst that may loop

We evaluated the efficiency of the mechanism on eight different service provider topologies (different network size, design). The benefit is displayed in the table above. The benefit is evaluated as follows:

- We consider a tuple (link A-B, destination D, PLR S, backup nexthop N) as a loop if upon link A-B failure, the flow from a router S upstream from A (A could be considered as PLR also) to D may loop due to convergence time difference between S and one of his neighbors N.

- We evaluate the number of potential loop tuples in normal conditions.

- We evaluate the number of potential loop tuples using the same topological input but taking into account that S converges after N.

- The gain is how many loops (both remote and local) we succeed to suppress.

On topology 1, 71% of the transient forwarding loops created by the failure of any link are prevented by implementing the local delay. The analysis shows that all local loops are prevented and only remote loops remain.

## 8. Deployment considerations

Transient forwarding loops have the following drawbacks:

- They limit FRR efficiency: even if FRR is activated within 50msec, as soon as PLR has converged, the traffic may be affected by a transient loop.

o They may impact traffic not directly affected by the failure (due to link congestion).

This local delay proposal is a transient forwarding loop avoidance mechanism (like OFIB). Even if it only addresses local transient loops, the efficiency versus complexity comparison of the mechanism makes it a good solution. It is also incrementally deployable with incremental benefits, which makes it an attractive option both for vendors to implement and service providers to deploy. Delaying the convergence time is not an issue if we consider that the traffic is protected during the convergence.

The ULOOP_DELAY_DOWN_TIMER value should be set according to the maximum IGP convergence time observed in the network (usually observed in the slowest node).

The proposed mechanism is limited to link down events. When a link goes down, it eventually goes back up. As a consequence, with the proposed mechanism deployed, only the link down event will be protected against transient forwarding loops while the link up event will not. If the operator wants to limit the impact of the transient forwarding loops during the link up event, it should take care of using specific procedures to bring the link back online. As examples, the operator can decide to put back the link online out of business hours or it can use some incremental metric changes to prevent loops (as proposed in [RFC5715]).

## 9. Examples

We will consider the following figure for the associated examples :

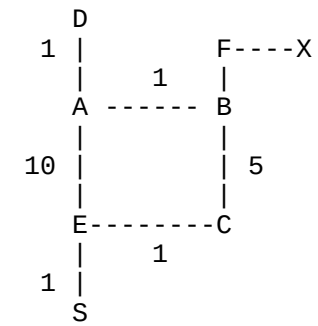



Figure 7

The network above is considered to have a convergence time about 1 second, so ULOOP_DELAY_DOWN_TIMER will be adjusted to this value. We also consider that FRR is running on each node.

### 9.1. Local link down

The table 3 describes the events and associated timing that happen on router C and E when link B-C goes down. It is based on a theoretical sequence of event that should only been read as an example. As C detects a single local event corresponding to a link down (its LSP + LSP from B received), it applies the local delay down behavior and no microloop is formed.

| Network condition | Time | Router C events | Router E events |
|---|---|---|---|
| S->D Traffic OK | | | |
| S->D Traffic lost | t0 | Link B-C fails | Link B-C fails |
| | t0+20msec | C detects the failure | |
| S->D Traffic OK | t0+40msec | C activates FRR | |
| | t0+50msec | C updates its local LSP/LSA | |
| | t0+60msec | C schedules SPF (100ms) | |
| | t0+67msec | C receives LSP/LSA from B | |
| | t0+70msec | C floods its local updated LSP/LSA | |
| | t0+87msec | | E receives LSP/LSA from C and schedules SPF (100ms) |
| | t0+117msec | | E floods LSP/LSA from C |
| | t0+160msec | C computes SPF | |
| | t0+165msec | C delays its RIB/FIB update (1 sec) | |
| | t0+193msec | | E computes SPF |
| | t0+199msec | | E starts updating |

| | | | |
|---|---|---|---|
| | | | its RIB/FIB |
| | t0+443msec | | E updates its RIB/FIB for D |
| | t0+470msec | | E convergence ends |
| | t0+1165msec | C starts updating its RIB/FIB | |
| | t0+1255msec | C updates its RIB/FIB for D | |
| | t0+1340msec | C convergence ends | |

Table 3 - Route computation event time scale

Similarly, upon B-C link down event, if LSP/LSA from B is received before C detects the link failure, C will apply the route update delay if the local detection is part of the same SPF run. The table 4 describes the associated theoretical sequence of events. It should only been read as an example.

| Network condition | Time | Router C events | Router E events |
|---|---|---|---|
| S->D Traffic OK | | | |
| S->D Traffic lost | t0 | Link B-C fails | Link B-C fails |
| | t0+32msec | C receives LSP/LSA from B | |
| | t0+33msec | C schedules SPF (100ms) | |
| | t0+50msec | C detects the failure | |
| S->D | t0+55msec | C activates FRR | |

| | | | |
|---|---|---|---|
| Traffic OK | | | |
| | t0+55msec | C updates its local LSP/LSA | |
| | t0+70msec | C floods its local updated LSP/LSA | |
| | t0+87msec | | E receives LSP/LSA from C and schedules SPF (100ms) |
| | t0+117msec | | E floods LSP/LSA from C |
| | t0+160msec | C computes SPF | |
| | t0+165msec | C delays its RIB/FIB update (1 sec) | |
| | t0+193msec | | E computes SPF |
| | t0+199msec | | E starts updating its RIB/FIB |
| | t0+443msec | | E updates its RIB/FIB for D |
| | t0+470msec | | E convergence ends |
| | t0+1165msec | C starts updating its RIB/FIB | |
| | t0+1255msec | C updates its RIB/FIB for D | |
| | t0+1340msec | C convergence ends | |

Table 4 - Route computation event time scale

### 9.2. Local and remote event

The table 5 describes the events and associated timing that happen on router C and E when link B-C goes down, in addition F-X link will fail in the same time window. C will not apply the local delay because a non local topology change is also received. The table 5 is based on a theoretical sequence of event that should only been read as an example.

| Network condition | Time | Router C events | Router E events |
|---|---|---|---|
| S->D Traffic OK | | | |
| S->D Traffic lost | t0 | Link B-C fails | Link B-C fails |
| | t0+20msec | C detects the failure | |
| | t0+36msec | Link F-X fails | Link F-X fails |
| S->D Traffic OK | t0+40msec | C activates FRR | |
| | t0+50msec | C updates its local LSP/LSA | |
| | t0+54msec | C receives LSP/LSA from F and floods it | |
| | t0+60msec | C schedules SPF (100ms) | |
| | t0+67msec | C receives LSP/LSA from B | |
| | t0+69msec | | E receives LSP/LSA from F, floods it and schedules SPF (100ms) |
| | t0+70msec | C floods its | |

| | | | | |
|---|---|---|---|---|
| | | local updated LSP/LSA | | |
| | t0+87msec | | E receives LSP/LSA from C | |
| | t0+117msec | | E floods LSP/LSA from C | |
| | t0+160msec | C computes SPF | | |
| | t0+165msec | C starts updating its RIB/FIB (NO DELAY) | | |
| | t0+170msec | | E computes SPF | |
| | t0+173msec | | E starts updating its RIB/FIB | |
| S->D Traffic lost | t0+365msec | C updates its RIB/FIB for D | | |
| S->D Traffic OK | t0+443msec | | E updates its RIB/FIB for D | |
| | t0+450msec | C convergence ends | | |
| | t0+470msec | | E convergence ends | |

Table 5 - Route computation event time scale

## 9.3. Aborting local delay

The table 6 describes the events and associated timing that happen on router C and E when link B-C goes down. In addition, we consider what happens when F-X link fails during local delay of the FIB update. C will first apply the local delay, but when the new event happens, it will fall back to the standard convergence mechanism without further delaying route insertion. In this example, we consider a ULOOP_DELAY_DOWN_TIMER configured to 2 seconds. The table

6 is based on a theoretical sequence of event that should only been read as an example.

| Network condition | Time | Router C events | Router E events |
|---|---|---|---|
| S->D Traffic OK | | | |
| S->D Traffic lost | t0 | Link B-C fails | Link B-C fails |
| | t0+20msec | C detects the failure | |
| S->D Traffic OK | t0+40msec | C activates FRR | |
| | t0+50msec | C updates its local LSP/LSA | |
| | t0+60msec | C schedules SPF (100ms) | |
| | t0+67msec | C receives LSP/LSA from B | |
| | t0+70msec | C floods its local updated LSP/LSA | |
| | t0+87msec | | E receives LSP/LSA from C and schedules SPF (100ms) |
| | t0+117msec | | E floods LSP/LSA from C |
| | t0+160msec | C computes SPF | |
| | t0+165msec | C delays its RIB/FIB update (2 sec) | |
| | t0+193msec | | E computes SPF |
| | t0+199msec | | E starts updating |

| | | | |
|---|---|---|---|
| | | | its RIB/FIB |
| | t0+254msec | Link F-X fails | Link F-X fails |
| | t0+300msec | C receives LSP/LSA from F and floods it | |
| | t0+303msec | C schedules SPF (200ms) | |
| | t0+312msec | E receives LSP/LSA from F and floods it | |
| | t0+313msec | E schedules SPF (200ms) | |
| | t0+502msec | C computes SPF | |
| | t0+505msec | C starts updating its RIB/FIB (NO DELAY) | |
| | t0+514msec | | E computes SPF |
| | t0+519msec | | E starts updating its RIB/FIB |
| S->D Traffic lost | t0+659msec | C updates its RIB/FIB for D | |
| S->D Traffic OK | t0+778msec | | E updates its RIB/FIB for D |
| | t0+781msec | C convergence ends | |
| | t0+810msec | | E convergence ends |

Table 6 - Route computation event time scale

## 10. Comparison with other solutions

As stated in Section 4, the proposed solution reuses some concepts already introduced by other IETF proposals but tries to find a tradeoff between efficiency and simplicity. This section tries to compare behaviors of the solutions.

### 10.1. PLSN

PLSN ([I-D.ietf-rtgwg-microloop-analysis]) describes a mechanism where each node in the network tries to avoid transient forwarding loops upon a topology change by always keeping traffic on a loop-free path for a defined duration (locked path to a safe neighbor). The locked path may be the new primary nexthop, another neighbor, or the old primary nexthop depending how the safety condition is satisfied.

PLSN does not solve all transient forwarding loops (see [I-D.ietf-rtgwg-microloop-analysis] Section 4 for more details).

Our solution reuses some concept of PLSN but in a more simple fashion:

- PLSN has three different behaviors: keep using old nexthop, use new primary nexthop if it is safe, or use another safe nexthop, while the proposed solution only has one: keep using the current nexthop (old primary, or already activated FRR path).

- PLSN may cause some damage while using a safe nexthop which is not the new primary nexthop in case the new safe nexthop does not provide enough bandwidth (see [RFC7916]). This solution may not experience this issue as the service provider may have control on the FRR path being used preventing network congestion.

- PLSN applies to all nodes in a network (remote or local changes), while the proposed mechanism applies only on the nodes connected to the topology change.

### 10.2. OFIB

OFIB ([RFC6976]) describes a mechanism where the convergence of the network upon a topology change is ordered in order to prevent transient forwarding loops. Each router in the network must deduce the failure type from the LSA/LSP received and computes/applies a specific FIB update timer based on the failure type and its rank in the network considering the failure point as root.

This mechanism allows to solve all the transient forwarding loop in a network at the price of introducing complexity in the convergence process that may require a strong monitoring by the service provider.

Our solution reuses the OFIB concept but limits it to the first hop that experiences the topology change. As demonstrated, the mechanism proposed in this document allows to solve all the local transient forwarding loops that represents an high percentage of all the loops. Moreover limiting the mechanism to one hop allows to keep the network-wide convergence behavior.

## 11. Implementation Status

At this time, there are three different implementations of this mechanism.

- Implementation 1:
  - Organization: Cisco
  - Implementation name: Local Microloop Protection
  - Operating system: IOS-XE
  - Level of maturity: production release
  - Coverage: all the specification is implemented
  - Protocols supported: ISIS and OSPF
  - Implementation experience: tested in lab and works as expected
  - Comment: the feature gives the ability to choose to apply the delay to FRR protected entry only
  - Report last update: 10-11-2017
- Implementation 2:
  - Organization: Cisco
  - Implementation name: Local Microloop Protection
  - Operating system: IOS-XR
  - Level of maturity: deployed
  - Coverage: all the specification is implemented

    * Protocols supported: ISIS and OSPF

    * Implementation experience: deployed and works as expected

    * Comment: the feature gives the ability to choose to apply the delay to FRR protected entry only

    * Report last update: 10-11-2017

  o Implementation 3:

    * Organization: Juniper Networks

    * Implementation name: Microloop avoidance when IS-IS link fails

    * Operating system: JUNOS

    * Level of maturity: deployed (hidden command)

    * Coverage: all the specification is implemented

    * Protocols supported: ISIS only

    * Implementation experience: deployed and works as expected

    * Comment: the feature applies to all the ISIS routes

    * Report last update: 10-11-2017

## 12. Security Considerations

This document does not introduce any change in term of IGP security. The operation is internal to the router. The local delay does not increase the number of attack vectors as an attacker could only trigger this mechanism if he already has be ability to disable or enable an IGP link. The local delay does not increase the negative consequences. If an attacker has the ability to disable or enable an IGP link, it can already harm the network by creating instability and harm the traffic by creating forwarding packet loss and forwarding loss for the traffic crossing that link.

## 13. Acknowledgements

We would like to thanks the authors of [RFC6976] for introducing the concept of ordered convergence: Mike Shand, Stewart Bryant, Stefano Previdi, and Olivier Bonaventure.

## 14. IANA Considerations

This document has no actions for IANA.

## 15. References

### 15.1. Normative References

[ISO10589] "Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)", ISO 10589, 2002.

[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>.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <https://www.rfc-editor.org/info/rfc2328>.

### 15.2. Informative References

[I-D.ietf-rtgwg-backoff-algo] Decraene, B., Litkowski, S., Gredler, H., Lindem, A., Francois, P., and C. Bowers, "SPF Back-off algorithm for link state IGPs", draft-ietf-rtgwg-backoff-algo-06 (work in progress), October 2017.

[I-D.ietf-rtgwg-microloop-analysis] Zinin, A., "Analysis and Minimization of Microloops in Link-state Routing Protocols", draft-ietf-rtgwg-microloop-analysis-01 (work in progress), October 2005.

[RFC3906] Shen, N. and H. Smit, "Calculating Interior Gateway Protocol (IGP) Routes Over Traffic Engineering Tunnels", RFC 3906, DOI 10.17487/RFC3906, October 2004, <https://www.rfc-editor.org/info/rfc3906>.

[RFC5715] Shand, M. and S. Bryant, "A Framework for Loop-Free Convergence", RFC 5715, DOI 10.17487/RFC5715, January 2010, <https://www.rfc-editor.org/info/rfc5715>.

[RFC6976] Shand, M., Bryant, S., Previdi, S., Filsfils, C., Francois, P., and O. Bonaventure, "Framework for Loop-Free Convergence Using the Ordered Forwarding Information Base (oFIB) Approach", RFC 6976, DOI 10.17487/RFC6976, July 2013, <https://www.rfc-editor.org/info/rfc6976>.

[RFC7916] Litkowski, S., Ed., Decraene, B., Filsfils, C., Raza, K., Horneffer, M., and P. Sarkar, "Operational Management of Loop-Free Alternates", RFC 7916, DOI 10.17487/RFC7916, July 2016, <https://www.rfc-editor.org/info/rfc7916>.

## Authors' Addresses

Stephane Litkowski
Orange

Email: stephane.litkowski@orange.com

Bruno Decraene
Orange

Email: bruno.decraene@orange.com

Clarence Filsfils
Cisco Systems

Email: cfilsfil@cisco.com

Pierre Francois
Individual

Email: pfrpfr@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 31, 2017

A. Lindem, Ed.
Cisco Systems
Y. Qu
Huawei
D. Yeung
Arrcus, Inc
I. Chen
Jabil
J. Zhang
Juniper Networks

April 29, 2017

# Routing Key Chain YANG Data Model
# draft-ietf-rtgwg-yang-key-chain-24.txt

## Abstract

This document describes the key chain YANG data model. Key chains are commonly used for routing protocol authentication and other applications requiring symmetric keys. A key chain is a list of elements each containing a key string, send lifetime, accept lifetime, and algorithm (authentication or encryption). By properly overlapping the send and accept lifetimes of multiple key chain elements, key strings and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated.

## Status of This Memo

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

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

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

This Internet-Draft will expire on October 31, 2017.

Table of Contents



## 1. Introduction

This document describes the key chain YANG [YANG-1.1] data model. Key chains are commonly used for routing protocol authentication and other applications requiring symmetric keys. A key chain is a list of elements each containing a key string, send lifetime, accept lifetime, and algorithm (authentication or encryption). By properly

overlapping the send and accept lifetimes of multiple key chain elements, key strings and algorithms may be gracefully updated. By representing them in a YANG data model, key distribution can be automated.

In some applications, the protocols do not use the key chain element key directly, but rather a key derivation function is used to derive a short-lived key from the key chain element key (e.g., the Master Keys used in [TCP-AO]).

### 1.1. Requirements Notation

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

### 1.2. Tree Diagrams

A simplified graphical representation of the complete data tree is presented in Section 3.3. The following tree notation is used.

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

## 2. Problem Statement

This document describes a YANG [YANG-1.1] data model for key chains. Key chains have been implemented and deployed by a large percentage of network equipment vendors. Providing a standard YANG model will facilitate automated key distribution and non-disruptive key

rollover. This will aid in tightening the security of the core routing infrastructure as recommended in [IAB-REPORT].

A key chain is a list containing one or more elements containing a Key ID, key string, send/accept lifetimes, and the associated authentication or encryption algorithm. A key chain can be used by any service or application requiring authentication or encryption using symmetric keys. In essence, the key-chain is a reusable key policy that can be referenced wherever it is required. The key-chain construct has been implemented by most networking vendors and deployed in many networks.

A conceptual representation of a crypto key table is described in [CRYPTO-KEYTABLE]. The crypto key table also includes keys as well as their corresponding lifetimes and algorithms. Additionally, the key table includes key selection criteria and envisions a deployment model where the details of the applications or services requiring authentication or encryption permeate into the key database. The YANG key-chain model described herein doesn’t include key selection criteria or support this deployment model. At the same time, it does not preclude it. The draft [YANG-CRYPTO-KEYTABLE] describes augmentations to the key chain YANG model in support of key selection criteria.

## 2.1. Applicability

Other YANG modules may reference ietf-key-chain YANG module key-chain names for authentication and encryption applications. A YANG type has been provided to facilitate reference to the key-chain name without having to specify the complete YANG XML Path Language (XPath) selector.

## 2.2. Graceful Key Rollover using Key Chains

Key chains may be used to gracefully update the key string and/or algorithm used by an application for authentication or encryption. To achieve graceful key rollover, the receiver MAY accept all the keys that have a valid accept lifetime and the sender MAY send the key with the most recent send lifetime. One scenario for facilitating key rollover is to:

1. Distribute a key chain with a new key to all the routers or other network devices in the domain of that key chain. The new key’s accept lifetime should be such that it is accepted during the key rollover period. The send lifetime should be a time in the future when it can be assured that all the routers in the domain of that key are upgraded. This will have no immediate impact on the keys used for transmission.

2. Assure that all the network devices have been updated with the updated key chain and that their system times are roughly synchronized. The system times of devices within an administrative domain are commonly synchronized (e.g., using Network Time Protocol (NTP) [NTP-PROTO]). This also may be automated.

3. When the send lifetime of the new key becomes valid, the network devices within the domain of key chain will using the new key for transmissions.

4. At some point in the future, a new key chain with the old key removed may be distributed to the network devices within the domain of the key chain. However, this may be deferred until the next key rollover. If this is done, the key chain will always include two keys; either the current and future key (during key rollovers) or the current and previous keys (between key rollovers).

Since the most recent send lifetime is defined as the one with the latest start-time, specification of "always" will prevent using the graceful key rollover technique described above. Other key configuration and usage scenarios are possible but these are beyond the scope of this document.

## 3. Design of the Key Chain Model

The ietf-key-chain module contains a list of one or more keys indexed by a Key ID. For some applications (e.g., OSPFv3 [OSPFV3-AUTH]), the Key ID is used to identify the key chain key to be used. In addition to the Key ID, each key chain key includes a key-string and a cryptographic algorithm. Optionally, the key chain keys include send/accept lifetimes. If the send/accept lifetime is unspecified, the key is always considered valid.

Note that different key values for transmission versus acceptance may be supported with multiple key chain elements. The key used for transmission will have a valid send-lifetime and invalid accept-lifetime (e.g., has an end-time equal to the start-time). The key used for acceptance will have a valid accept-lifetime and invalid send-lifetime.

Due to the differences in key chain implementations across various vendors, some of the data elements are optional. Finally, the crypto algorithm identities are provided for reuse when configuring legacy authentication and encryption not using key-chains.

A key-chain is identified by a unique name within the scope of the network device. The "key-chain-ref" typedef SHOULD be used by other YANG modules when they need to reference a configured key-chain.

### 3.1. Key Chain Operational State

The key chain operational state is included in the same tree as key chain configuration consistent with Network Management Datastore Architecture [NMDA]. The timestamp of the last key chain modification is also maintained in the operational state. Additionally, the operational state includes an indication of whether or not a key chain key is valid for sending or acceptance.

### 3.2. Key Chain Model Features

Features are used to handle differences between vendor implementations. For example, not all vendors support configuration of an acceptance tolerance or configuration of key strings in hexadecimal. They are also used to support of security requirements (e.g., TCP-AO Algorithms [TCP-AO-ALGORITHMS]) not yet implemented by vendors or only a single vendor.

It is common for an entity with sufficient permissions to read and store a device’s configuration which would include the contents of this model. To avoid unnecessarily seeing and storing the keys in clear-text, this model provides the aes-key-wrap feature. More details are described in Security Considerations Section 5.

### 3.3. Key Chain Model Tree

```
+--rw key-chains
   +--rw key-chain* [name]
   |  +--rw name                        string
   |  +--rw description?                string
   |  +--rw accept-tolerance {accept-tolerance}?
   |  |  +--rw duration?   uint32
   |  +--ro last-modified-timestamp?   yang:date-and-time
   |  +--rw key* [key-id]
   |     +--rw key-id                     uint64
   |     +--rw lifetime
   |     |  +--rw (lifetime)?
   |     |     +--:(send-and-accept-lifetime)
   |     |     |  +--rw send-accept-lifetime
   |     |     |     +--rw (lifetime)?
   |     |     |        +--:(always)
   |     |     |        |  +--rw always?            empty
   |     |     |        +--:(start-end-time)
   |     |     |           +--rw start-date-time?
```

```
|     |     |           |       yang:date-and-time
|     |     |           +--rw (end-time)?
|     |     |              +--:(infinite)
|     |     |              |  +--rw no-end-time?       empty
|     |     |              +--:(duration)
|     |     |              |  +--rw duration?          uint32
|     |     |              +--:(end-date-time)
|     |     |                 +--rw end-date-time?
|     |     |                            yang:date-and-time
|     |     +--:(independent-send-accept-lifetime)
|     |     |   {independent-send-accept-lifetime}?
|     |     +--rw send-lifetime
|     |     |  +--rw (lifetime)?
|     |     |     +--:(always)
|     |     |     |  +--rw always?            empty
|     |     |     +--:(start-end-time)
|     |     |        +--rw start-date-time?
|     |     |        |       yang:date-and-time
|     |     |        +--rw (end-time)?
|     |     |           +--:(infinite)
|     |     |           |  +--rw no-end-time?       empty
|     |     |           +--:(duration)
|     |     |           |  +--rw duration?          uint32
|     |     |           +--:(end-date-time)
|     |     |              +--rw end-date-time?
|     |     |                         yang:date-and-time
|     |     |        +--rw accept-lifetime
|     |     |           +--rw (lifetime)?
|     |     |              +--:(always)
|     |     |              |  +--rw always?            empty
|     |     |              +--:(start-end-time)
|     |     |                 +--rw start-date-time?
|     |     |                 |       yang:date-and-time
|     |     |                 +--rw (end-time)?
|     |     |                    +--:(infinite)
|     |     |                    |  +--rw no-end-time?       empty
|     |     |                    +--:(duration)
|     |     |                    |  +--rw duration?          uint32
|     |     |                    +--:(end-date-time)
|     |     |                       +--rw end-date-time?
|     |     |                                  yang:date-and-time
|     |     +--rw crypto-algorithm identityref
|     |     +--rw key-string
|     |     |  +--rw (key-string-style)?
|     |     |     +--:(keystring)
|     |     |     |  +--rw keystring?            string
|     |     |     +--:(hexadecimal) {hex-key-string}?
|     |     |        +--rw hexadecimal-string?   yang:hex-string
```

```
      |     +--ro send-lifetime-active?     boolean
      |     +--ro accept-lifetime-active?   boolean
      +--rw aes-key-wrap {aes-key-wrap}?
         +--rw enable?   boolean
```

## 4. Key Chain YANG Model

```
<CODE BEGINS> file "ietf-key-chain@2017-04-18.yang"
module ietf-key-chain {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-key-chain";
  prefix key-chain;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-netconf-acm {
    prefix nacm;
  }

  organization
    "IETF RTG (Routing) Working Group";
  contact
    "Acee Lindem - acee@cisco.com";
  description
    "This YANG module defines the generic configuration
     data for key-chain. It is intended that the module
     will be extended by vendors to define vendor-specific
     key-chain configuration parameters.

     Copyright (c) 2017 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 RFC XXXX; see
     the RFC itself for full legal notices.";

  revision 2017-04-18 {
    description
      "Initial RFC Revision";
    reference "RFC XXXX: A YANG Data Model for key-chain";
  }
```

```
     feature hex-key-string {
       description
         "Support hexadecimal key string.";
     }

     feature accept-tolerance {
       description
         "Support the tolerance or acceptance limit.";
     }

     feature independent-send-accept-lifetime {
       description
         "Support for independent send and accept key lifetimes.";
     }

     feature crypto-hmac-sha-1-12 {
       description
         "Support for TCP HMAC-SHA-1 12 byte digest hack.";
     }

     feature clear-text {
       description
         "Support for clear-text algorithm. Usage is
          NOT RECOMMENDED.";
     }

     feature aes-cmac-prf-128 {
       description
         "Support for AES Cipher based Message Authentication
          Code Pseudo Random Function.";
     }

     feature aes-key-wrap {
       description
         "Support for Advanced Encryption Standard (AES) Key Wrap.";
     }

     feature replay-protection-only {
       description
         "Provide replay-protection without any authentication
          as required by protocols such as Bidirectional
          Forwarding Detection (BFD).";
     }
     identity crypto-algorithm {
       description
         "Base identity of cryptographic algorithm options.";
     }
```

```
     identity hmac-sha-1-12 {
       base crypto-algorithm;
       if-feature "crypto-hmac-sha-1-12";
       description
         "The HMAC-SHA1-12 algorithm.";
     }

     identity aes-cmac-prf-128 {
       base crypto-algorithm;
       if-feature "aes-cmac-prf-128";
       description
         "The AES-CMAC-PRF-128 algorithm - required by
          RFC 5926 for TCP-AO key derivation functions.";
     }

     identity md5 {
       base crypto-algorithm;
       description
         "The MD5 algorithm.";
     }

     identity sha-1 {
       base crypto-algorithm;
       description
         "The SHA-1 algorithm.";
     }

     identity hmac-sha-1 {
       base crypto-algorithm;
       description
         "HMAC-SHA-1 authentication algorithm.";
     }

     identity hmac-sha-256 {
       base crypto-algorithm;
       description
         "HMAC-SHA-256 authentication algorithm.";
     }

     identity hmac-sha-384 {
       base crypto-algorithm;
       description
         "HMAC-SHA-384 authentication algorithm.";
     }

     identity hmac-sha-512 {
       base crypto-algorithm;
       description
```

```
        "HMAC-SHA-512 authentication algorithm.";
    }

    identity clear-text {
      base crypto-algorithm;
      if-feature "clear-text";
      description
        "Clear text.";
    }

    identity replay-protection-only {
      base crypto-algorithm;
      if-feature "replay-protection-only";
      description
        "Provide replay-protection without any authentication as
         required by protocols such as Bidirectional Forwarding
         Detection (BFD).";
    }

    typedef key-chain-ref {
      type leafref {
        path
        "/key-chain:key-chains/key-chain:key-chain/key-chain:name";
      }
      description
        "This type is used by data models that need to reference
         configured key-chains.";
    }

    grouping lifetime {
      description
        "Key lifetime specification.";
      choice lifetime {
        default "always";
        description
          "Options for specifying key accept or send lifetimes";
        case always {
          leaf always {
            type empty;
            description
              "Indicates key lifetime is always valid.";
          }
        }
        case start-end-time {
          leaf start-date-time {
            type yang:date-and-time;
            description
              "Start time.";
```

```
          }
          choice end-time {
            default "infinite";
            description
              "End-time setting.";
            case infinite {
              leaf no-end-time {
                type empty;
                description
                  "Indicates key lifetime end-time is infinite.";
              }
            }
            case duration {
              leaf duration {
                type uint32 {
                  range "1..2147483646";
                }
                units "seconds";
                description
                  "Key lifetime duration, in seconds";
              }
            }
            case end-date-time {
              leaf end-date-time {
                type yang:date-and-time;
                description
                  "End time.";
              }
            }
          }
        }
      }
    }
    grouping key-common {
      description
        "Key-chain key data nodes common to
         configuration and state.";
    }

    container key-chains {
      description
        "All configured key-chains on the device.";
      list key-chain {
        key "name";
        description
          "List of key-chains.";
        leaf name {
          type string;
```

```
          description
            "Name of the key-chain.";
        }
        leaf description {
          type string;
          description
            "A description of the key-chain";
        }
        container accept-tolerance {
          if-feature "accept-tolerance";
          description
            "Tolerance for key lifetime acceptance (seconds).";
          leaf duration {
            type uint32;
            units "seconds";
            default "0";
            description
              "Tolerance range, in seconds.";
          }
        }
        leaf last-modified-timestamp {
          type yang:date-and-time;
          config false;
          description
            "Timestamp of the most recent update to the key-chain";
        }
        list key {
          key "key-id";
          description
            "Single key in key chain.";
          leaf key-id {
            type uint64;
            description
              "Numeric value uniquely identifying the key";
          }
          container lifetime {
            description
              "Specify a key's lifetime.";
            choice lifetime {
              description
                "Options for specification of send and accept
                 lifetimes.";
              case send-and-accept-lifetime {
                description
                  "Send and accept key have the same lifetime.";
                container send-accept-lifetime {
                  description
                    "Single lifetime specification for both
```

```
                    send and accept lifetimes.";
                uses lifetime;
              }
            }
            case independent-send-accept-lifetime {
              if-feature "independent-send-accept-lifetime";
              description
                "Independent send and accept key lifetimes.";
              container send-lifetime {
                description
                  "Separate lifetime specification for send
                   lifetime.";
                uses lifetime;
              }
              container accept-lifetime {
                description
                  "Separate lifetime specification for accept
                   lifetime.";
                uses lifetime;
              }
            }
          }
        }
        leaf crypto-algorithm {
          type identityref {
            base crypto-algorithm;
          }
          mandatory true;
          description
            "Cryptographic algorithm associated with key.";
        }
        container key-string {
          description
            "The key string.";
          nacm:default-deny-all;
          choice key-string-style {
            description
              "Key string styles";
             case keystring {
               leaf keystring {
                type string;
                description
                  "Key string in ASCII format.";
              }
            }
            case hexadecimal {
              if-feature "hex-key-string";
              leaf hexadecimal-string {
```

```
                    type yang:hex-string;
                    description
                      "Key in hexadecimal string format. When compared
                       to ASCII, specification in hexadecimal affords
                       greater key entropy with the same number of
                       internal key-string octets. Additionally, it
                       discourages usage of well-known words or
                       numbers.";
                  }
                }
              }
            }
            leaf send-lifetime-active {
              type boolean;
              config false;
              description
                "Indicates if the send lifetime of the
                 key-chain key is currently active.";
               }
            leaf accept-lifetime-active {
              type boolean;
              config false;
              description
                "Indicates if the accept lifetime of the
                 key-chain key is currently active.";
            }
          }
        }
        container aes-key-wrap {
          if-feature "aes-key-wrap";
          description
            "AES Key Wrap encryption for key-chain key-strings. The
             encrypted key-strings are encoded as hexadecimal key
             strings using the hex-key-string leaf.";
          leaf enable {
            type boolean;
            default "false";
            description
              "Enable AES Key Wrap encryption.";
          }
        }
      }
    }
    <CODE ENDS>
```

## 5. Security Considerations

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

The NETCONF access control model [NETCONF-ACM] provides the means to restrict access for particular NETCONF or RESTCONF users to a pre-configured subset of all available NETCONF or RESTCONF protocol operations and content. The key strings are not accessible by default and NETCONF Access Control Mode [NETCONF-ACM] rules are required to configure or retrieve them.

When configured, the key-strings can be encrypted using the AES Key Wrap algorithm [AES-KEY-WRAP]. The AES key-encryption key (KEK) is not included in the YANG model and must be set or derived independent of key-chain configuration. When AES key-encryption is used, the hex-key-string feature is also required since the encrypted keys will contain characters that are not representable in the YANG string built-in type [YANG-1.1]. It is RECOMMENDED that key-strings be encrypted using AES key-encryption to prevent key-chains from being retrieved and stored with the key-strings in clear text. This recommendation is independent of the access protection that is availed from the NETCONF Access Control Model (NACM) [NETCONF-ACM].

The clear-text algorithm is included as a YANG feature. Usage is NOT RECOMMENDED except in cases where the application and device have no other alternative (e.g., a legacy network device that must authenticate packets at intervals of 10 milliseconds or less for many peers using Bidirectional Forwarding Detection [BFD]). Keys used with the clear-text algorithm are considered insecure and SHOULD NOT be reused with more secure algorithms.

Similarly, the MD5 and SHA-1 algorithms have been proven to be insecure ([Dobb96a], [Dobb96b], and [SHA-SEC-CON]) and usage is NOT RECOMMENDED. Usage should be confined to deployments where it is required for backward compatibility.

Implementations with keys provided via this model should store them using best current security practices.

## 6. IANA Considerations

This document registers a URI in the IETF XML registry [XML-REGISTRY]. Following the format in [XML-REGISTRY], the following registration is requested to be made:

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

This document registers a YANG module in the YANG Module Names registry [YANG-1.0].

name: ietf-key-chain
namespace: urn:ietf:params:xml:ns:yang:ietf-key-chain
prefix: key-chain
reference: RFC XXXX

## 7. Contributors

Contributors' Addresses

Yi Yang
SockRate

Email: yi.yang@sockrate.com

## 8. References

### 8.1. Normative References

[NETCONF] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[NETCONF-ACM] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[RFC-KEYWORDS] Bradner, S., "Key words for use in RFC's to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[XML-REGISTRY] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[YANG-1.0]
Bjorklund, M., "YANG - A Data Modeling Language for Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[YANG-1.1]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, August 2016.

## 8.2. Informative References

[AES-KEY-WRAP]
Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 5649, August 2009.

[BFD] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, June 2010.

[CRYPTO-KEYTABLE]
Housley, R., Polk, T., Hartman, S., and D. Zhang, "Table of Cryptographic Keys", RFC 7210, April 2014.

[Dobb96a] Dobbertin, H., "Cryptanalysis of MD5 Compress", Technical Report (Presented at the RUMP Session of EuroCrypt 1996), 2 May 1996.

[Dobb96b] Dobbertin, H., "The Status of MD5 After a Recent Attack", CryptoBytes Vol. 2, No. 2, Summer 1996.

[IAB-REPORT]
Andersson, L., Davies, E., and L. Zhang, "Report from the IAB workshop on Unwanted Traffic March 9-10, 2006", RFC 4948, August 2007.

[NETCONF-SSH]
Wasserman, M., "NETCONF over SSH", RFC 6242, June 2011.

[NMDA] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watson, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-01.txt (work in progress), March 2017.

[NTP-PROTO]
Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[OSPFV3-AUTH]
Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, March 2014.

[RESTCONF]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, January 2017.

[SHA-SEC-CON]
Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, February 2011.

[TCP-AO] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[TCP-AO-ALGORITHMS]
Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.

[TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol", RFC 5246, August 2008.

[YANG-CRYPTO-KEYTABLE]
Chen, I., "YANG Data Model for RFC 7210 Key Table", draft-chen-rtg-key-table-yang-00.txt (work in progress), November 2015.

## Appendix A. Examples

### A.1. Simple Key Chain with Always Valid Single Key

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <key-chains xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
    <key-chain>
      <name>keychain-no-end-time</name>
      <description>
        A key chain with a single key that is always valid for
        transmission and reception.
      </description>
      <key>
        <key-id>100</key-id>
        <lifetime>
          <send-accept-lifetime>
            <always/>
          </send-accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-256</crypto-algorithm>
        <key-string>
          <keystring>keystring_in_ascii_100</keystring>
        </key-string>
      </key>
    </key-chain>
  </key-chains>
</data>
```

### A.2. Key Chain with Keys having Different Lifetimes

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <key-chains xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
    <key-chain>
      <name>keychain2</name>
      <description>
        A key chain where each key contains different send time
        and accept time and a different algorithm illustrating
        algorithm agility.
      </description>
      <key>
        <key-id>35</key-id>
        <lifetime>
          <send-lifetime>
            <start-date-time>2017-01-01T00:00:00Z</start-date-time>
            <end-date-time>2017-02-01T00:00:00Z</end-date-time>
          </send-lifetime>
          <accept-lifetime>
            <start-date-time>2016-12-31T23:59:55Z</start-date-time>
            <end-date-time>2017-02-01T00:00:05Z</end-date-time>
          </accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-256</crypto-algorithm>
        <key-string>
          <keystring>keystring_in_ascii_35</keystring>
        </key-string>
      </key>
      <key>
        <key-id>36</key-id>
        <lifetime>
          <send-lifetime>
            <start-date-time>2017-02-01T00:00:00Z</start-date-time>
            <end-date-time>2017-03-01T00:00:00Z</end-date-time>
          </send-lifetime>
          <accept-lifetime>
            <start-date-time>2017-01-31T23:59:55Z</start-date-time>
            <end-date-time>2017-03-01T00:00:05Z</end-date-time>
          </accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-512</crypto-algorithm>
        <key-string>
          <hexadecimal-string>fe:ed:be:af:36</hexadecimal-string>
        </key-string>
      </key>
    </key-chain>
  </key-chains>
</data>
```

### A.3. Key Chain with Independent Send and Accept Lifetimes

```
<?xml version="1.0" encoding="utf-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <key-chains xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
    <key-chain>
      <name>keychain2</name>
      <description>
        A key chain where each key contains different send time
        and accept times.
      </description>
      <key>
        <key-id>35</key-id>
        <lifetime>
          <send-lifetime>
            <start-date-time>2017-01-01T00:00:00Z</start-date-time>
            <end-date-time>2017-02-01T00:00:00Z</end-date-time>
          </send-lifetime>
          <accept-lifetime>
            <start-date-time>2016-12-31T23:59:55Z</start-date-time>
            <end-date-time>2017-02-01T00:00:05Z</end-date-time>
          </accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-256</crypto-algorithm>
        <key-string>
          <keystring>keystring_in_ascii_35</keystring>
        </key-string>
      </key>
      <key>
        <key-id>36</key-id>
        <lifetime>
          <send-lifetime>
            <start-date-time>2017-02-01T00:00:00Z</start-date-time>
            <end-date-time>2017-03-01T00:00:00Z</end-date-time>
          </send-lifetime>
          <accept-lifetime>
            <start-date-time>2017-01-31T23:59:55Z</start-date-time>
            <end-date-time>2017-03-01T00:00:05Z</end-date-time>
          </accept-lifetime>
        </lifetime>
        <crypto-algorithm>hmac-sha-256</crypto-algorithm>
        <key-string>
          <hexadecimal-string>fe:ed:be:af:36</hexadecimal-string>
        </key-string>
      </key>
    </key-chain>
  </key-chains>
</data>
```

## Appendix B. Acknowledgments

The RFC text was produced using Marshall Rose's xml2rfc tool.

Thanks to Brian Weis for fruitful discussions on security requirements.

Thanks to Ines Robles for Routing Directorate QA review comments.

Thanks to Ladislav Lhotka for YANG Doctor comments.

Thanks to Martin Bjorklund for additional YANG Doctor comments.

Thanks to Tom Petch for comments during IETF last call.

Thanks to Matthew Miller for comments made during the Gen-ART review.

Thanks to Vincent Roca for comments made during the Security Directorate review.

Thanks to Warren Kumari, Ben Campbell, Adam Roach, and Benoit Claise for comments received during the IESG review.

## Authors' Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Yingzhen Qu
Huawei

Email: yingzhen.qu@huawei.com

Derek Yeung
Arrcus, Inc

Email: derek@arrcus.com

Ing-Wher Chen
Jabil

Email: ing-wher_chen@jabil.com

Jeffrey Zhang
Juniper Networks
10 Technology Park Drive
Westford, MA 01886
USA

Email: zzhang@juniper.net

Inter-Domain Routing
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

P. Lapukhov
Facebook

October 31, 2016

# Use of BGP for dissemination of ILA mapping information draft-lapukhov-bgp-ila-afi-02

## Abstract

Identifier-Locator Addressing [I-D.herbert-nvo3-ila] relies on splitting the 128-bit IPv6 address into identifier and locator parts to implement identifier mobility, and network virtualization. This document proposes a method for distributing the identifier to locator mapping information using Multiprotocol Extensions for BGP-4 [RFC4760].

## Requirements Language

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

## Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

## Copyright Notice

Table of Contents



## 1. Introduction

Under the ILA proposal, IPv6 address is split in 64-bit identifier (lower address bits) and locator (higher address bits) portions. The locator part is determined dynamically from a mapping table that maintains associations between the location-independent identifiers and topologically significant locators. The hosts that collectively implement and maintain such mappings are referred to as "ILA domain" in this document. The ILA domain has a globally unique 64-bit SIR (Standard Identifier Representation) prefix assigned to it (see [I-D.herbert-nvo3-ila] for more details on SIR prefix use).

This document proposes a new address family identifier (AFI) for the purpose of disseminating the locator-identifier mappings among the nodes participating in the ILA domain. For example, this extension could be used to propagate the mappings from ILA hosts to ILA routers and allow the routers to perform their function (see

[I-D.herbert-nvo3-ila] for definition of the "ILA router" functions). Additional information that provides more detailed examples of deployment scenarios using BGP could be found in [I-D.lapukhov-ila-deployment].

## 2. BGP ILA AFI

This document introduces a new AFI known as a "Identifier-Locator Addressing AFI" (ILA AFI) with the actual value to be assigned by IANA. The purpose of this AFI is disseminating the mapping information in the ILA domain, e.g. between ILA hosts and ILA routers. This document defines the use of SAFI values of "1" (unicast) and "2" (multicast) only.

## 3. Capability Advertisement

A BGP speaker that wishes to exchange ILA mapping information MUST use the Multiprotocol Extensions Capability Code, as defined in [RFC4760], to advertise the corresponding AFI/SAFI pair.

## 4. Disseminating Identifier-Locator mapping information

### 4.1. Advertising ILA mapping information

For the purpose of ILA mapping encoding, the 8-octet locator field SHALL be encoded in the "next-hop address" field. The "length of the next-hop address" MUST be set to "8". The identifiers bound to the locator SHALL be encoded within the NLRI portion of MP_REACH_NLRI attribute. The NLRI portion of MP_REACH_NLRI starts with the two-octet "Length of identifiers" field, with the value being multiple of 8. The rest of the NLRI is a collection of 8-octet identifiers that are bound to the locator specified in the "next-hop address" field.

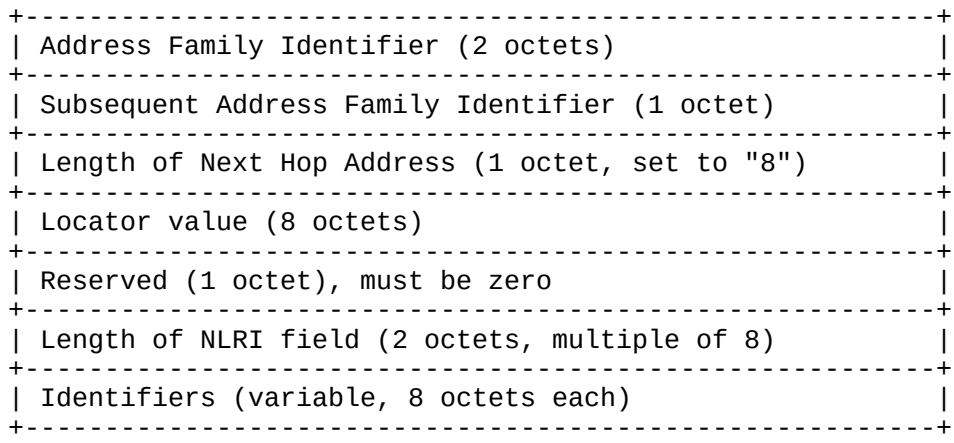



Figure 1: MP_REACH_NLRI Layout

### 4.2. Withdrawing ILA mapping information

Withdrawal of ILA mapping information is performed via an MP_UNREACH_NLRI attribute advertisement organized as following:

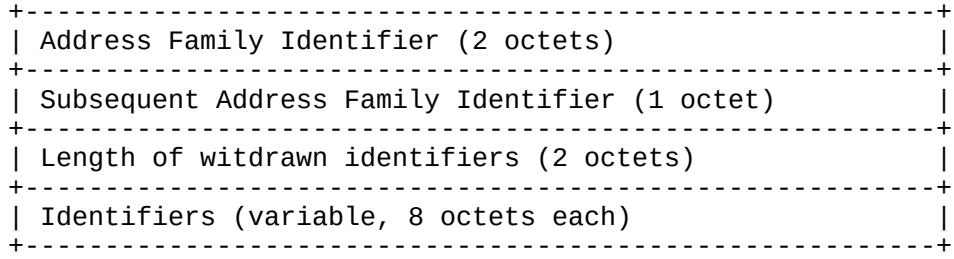



Figure 2: MP_UNREACH_NLRI Layout

## 5. Interpreting the mapping information

### 5.1. Unicast SAFI

Only the locator part of ILA address is used for packet routing, and every node that hosts an identifier is expected to have a unique /64 prefix routable within the scope of the ILA domain. The identifiers advertised under the ILA AFI are expected to be used by the data-plane implementation to perform match on a full IPv6 address and decide whether the locator portion of the address needs a re-write. It is up to the implementation to decide which full 128-bit IPv6 addresses need a rewrite, e.g. by matching on a Standard Identifier Representation (SIR) prefix as defined in [I-D.herbert-nvo3-ila].

The locator rewrite information comes from the next-hop "address" associated with the identifier. The next-hop field of MP_REACH_NLRI attribute in general is not expected to be used for any routing resolutions/lookups by the BGP process. It should primarily be used to create a rewrite rule in the data-plane forwarding table. The actual forwarding decision is then based on subsequent lookup in the forwarding table to find the next hop to send the packet to.

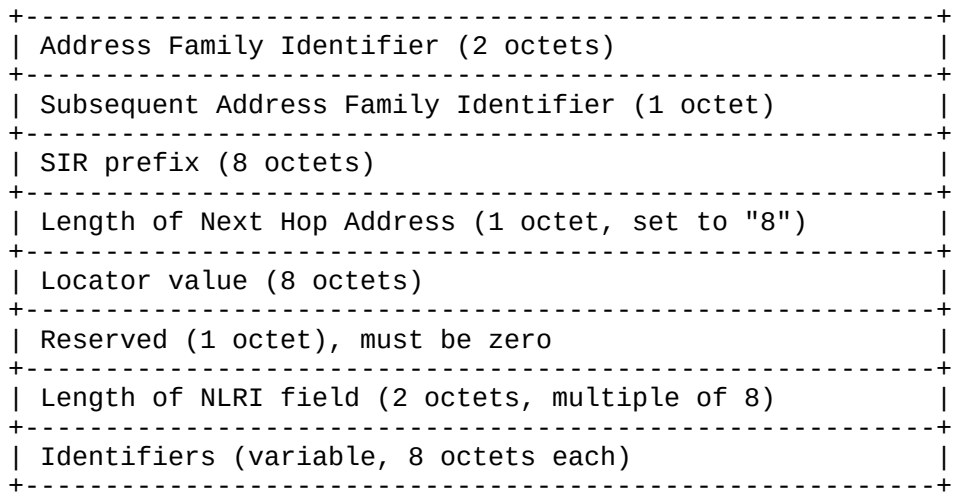
In some scenarios, resolving the next-hop attribute via additional lookups tables might be necessary. For example, for environments that deploy MPLS ([RFC3031]) forwarding, the locator may resolve to a label stack that is required to perform further forwarding. In this case, the ILA address rewrite will be accompanied by additional actions, such as label stack imposition. These decisions have to be made in implementation dependant fashion.

### 5.2. Multicast SAFI

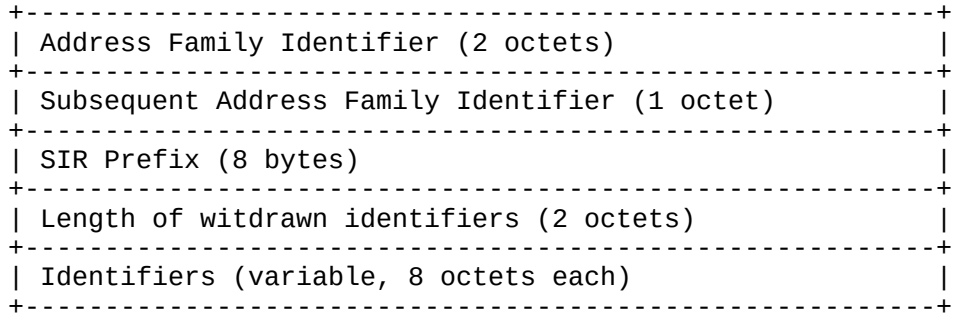
Multicast SAFI retain the same encoding format, with a different SAFI value. For multicast packets, the RPF check process SHALL be modified for use with ILA source addresses. Specifically, source ILA IPv6 addresses with the identifier portion matching the mapping table SHALL be mapped to proper locator, prior to performing the RPF check. The ILA source addresses need to be identified by some means specific to ILA implementation, e.g. by matching on configured SIR prefixes. The ILA addresses that do not match any mapping entry SHALL be considered as failing the RPF check.

## 6. Inter-domain mapping exchange

The ILA mappings are only unique with an ILA domain - it is possible that different domains may re-use the same identifiers. To make identifiers globally unique, they MUST be concatenated with the SIR prefix assigned to each ILA domain. These globally unique identifiers may then be exchanged between multiple ILA domains. IANA will be requested to allocate new SAFI value called "VPN-ILA" SAFI to facilitate exchange of inter-domain ILA mappings. The MP_REACH_NLRI attributes exchanged over this SAFI will look as following:




The main difference from the Unicast/Multicast SAFI's is presence of the SIR prefix field in the announcement. Correspondingly, the MP_UNREACH_NLRI will look as following:




The use of domain-specific identifiers would require the ILA hosts and routers to make their ILA cache lookups based on the full 128-bit prefix.

## 7. BGP Next-Hop attribute handling with ILA

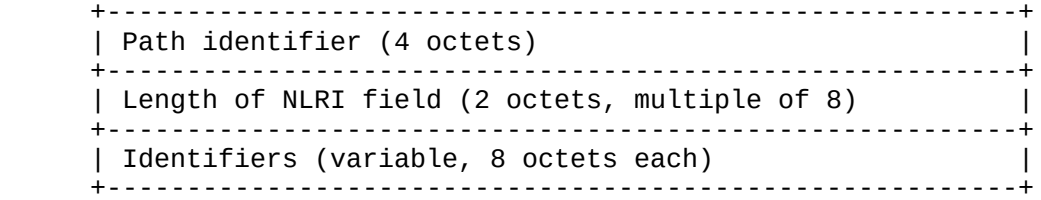
This document proposes that the BGP next-hop attribute value encode the locator associated with all identifiers found in MP_REACH_NLRI attribute. It is possible that an intermediate speaker may change the next-hop value. This may be required to ensure all traffic for the associated identifiers is routed through that intermediate speaker. The speaker is expected to maintain the original ILA mappings in its mapping table, and perform additional destination

address translation for the ILA packets. This way, a form of loose hop traffic engineering could be realized within an ILA domain.

It is common that BGP implementations reset the next-hop value for announcements made over eBGP sessions. Such scenario may be common between two different ILA domains.

## 8. Use of Add-Paths extension with ILA AFI

It could be useful to bind multiple locators to the same identifier, e.g. for the purpose of load-sharing. To make announcing multiple locators possible, the MP_REACH_NLRI and MP_REACH_NLRI attributes are extended to encode the path-identifier per [I-D.ietf-idr-add-paths], correspondingly enabling the capability for the AFI/SAFI. Specifically, the NLRI field will look as following:




Such encoding instructs the receiver to create multiple locator entries for an identifier, differentiated by their path identifiers. Optionally, a weight could be associated with each path using the "link bandwidth" extended community defined in [I-D.ietf-idr-link-bandwidth]

## 9. IANA Considerations

For the purpose of this work, IANA would be asked to allocate a value for the new AFI, and the VPN-ILA SAFI.

## 10. Manageability Considerations

The ILA mappings distribution is likely to be done using separate infrastructure, independent from the BGP topology used for regular routing information distribution. Most likely there will be a collection of iBGP route-reflectors deployed within each ILA domain, and peering with a BGP process running on each ILA router and ILA host. More details and deployment examples could be found in [I-D.lapukhov-ila-deployment].

## 11. Security Considerations

This document does not introduce any changes in terms of BGP security. Defining ILA security model is outside of scope of this document.

## 12. Acknowledgements

The author would like to thank Doug Porter for the original idea and discussion of this proposal.

## 13. References

### 13.1. Normative References

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

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <http://www.rfc-editor.org/info/rfc3031>.

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

[I-D.ietf-idr-add-paths] Walton, D., Retana, A., Chen, E., and J. Scudder, "Advertisement of Multiple Paths in BGP", draft-ietf-idr-add-paths-15 (work in progress), May 2016.

[I-D.ietf-idr-link-bandwidth] Mohapatra, P. and R. Fernando, "BGP Link Bandwidth Extended Community", draft-ietf-idr-link-bandwidth-06 (work in progress), January 2013.

### 13.2. Informative References

[RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", RFC 4760, DOI 10.17487/RFC4760, January 2007, <http://www.rfc-editor.org/info/rfc4760>.

[I-D.herbert-nvo3-ila]
Herbert, T., "Identifier-locator addressing for IPv6", draft-herbert-nvo3-ila-03 (work in progress), October 2016.

[I-D.lapukhov-ila-deployment]
Lapukhov, P., "Deploying Identifier-Locator Addressing (ILA) in datacenter", draft-lapukhov-ila-deployment-00 (work in progress), March 2016.

## Author's Address

Petr Lapukhov
Facebook
1 Hacker Way
Menlo Park, CA 94025
US

Email: petr@fb.com

opsawg
Internet-Draft
Intended status: Standards Track
Expires: September 19, 2016

P. Lapukhov
Facebook

March 18, 2016

# Data-plane probe for in-band telemetry collection
# draft-lapukhov-dataplane-probe-00

## Abstract

Detecting and isolating network faults in IP networks has traditionally been done using tools like ping and traceroute (see [RFC7276]) or more complex systems built on similar concepts of active probing and path tracing. While using active synthetic probes is proven to be helpful in detecting data-plane faults, isolating fault location has proven to be a much harder problem, especially in diverse networks with multiple active forwarding planes (e.g. IP and MPLS). Moreover, existing end-to-end tools do not generally support functionality beyond dealing with packet loss - for example, they are hardly useful for detecting and reporting transient (i.e. milli- or even micro-second) network congestion.

Modern network forwarding hardware can enable more sophisticated data-plane functionality that provides substantial improvement to the isolation and identification capabilities of network elements. For example, it has become possible to encode a snapshot of a network elements forwarding state within the packet payload as it transits the device. One example of such device/network state would be queue depth on the egress port taken by that specific packet. When combined with a unique device identifier embedded in the same packet, this could allow for precise time and topological identification of the the congested location within the network.

This document proposes a standard format for embedding telemetry information in UDP-based probing packets, i.e. packets designated for testing the network while not carrying application traffic. These active probes could be conveyed over multiple protocols (ICMP, UDP, TCP, etc.) but this document specifically focuses on UDP, given its simple semantics. In addition this document provides recommendations on handling the active probes by devices that do not support the required data-plane functionality.

## Status of This Memo

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

## Copyright Notice



## Table of Contents

## 1. Introduction

Detecting and isolating faults in IP networks may involve multiple tools and approaches, but by far the two most popular utilities used by operators are ping and traceroute. The ping utility provides the basic end-to-end connectivity check by sending a special ICMP packet. There are other variants of ping that work using TCP or UDP probes, but may require a special responder application (for UDP) on the other end of the probed connection.

This type of active probing approach has its limitations. First, it operates end-to-end and thus it is impossible to tell where in the path the fault has happened from simply observing the packet loss ratios. Secondly, in multipath (ECMP) scenarios it can be quite difficult to fully and/or deterministically exercise all the possible paths connecting two end-points.

The traceroute utility has multiple variants as well - UDP, ICMP and TCP based, for instance, and special variant for MPLS LSP testing. Practically all variants follow the same model of operations: varying TTL field setting in outgoing probes and analyzing the returned ICMP unreachable messages. This does allow isolating the fault down to the IP hop that is losing packets, but has its own limitations. As with the ping utility, it becomes complicated to explore all possible ECMP paths in the network. This is especially problematic in large Clos fabric topologies that are very common in large data-center networks. Next, many network devices limit the rate of outgoing ICMP messages as well as the rate of "exception" packets "punted" to the control plane processor. This puts a functional limit on the packet rate that the traceroute can probe a given hop with, and hence impacts the resolution and time to isolate a fault. Lastly, the treatment for these control packets is often different from the packets that take regular forwarding path: the latter are normally not redirected to the control plane processor and handled purely in the data-plane hardware.

Modern network processing elements (both hardware and software based) are capable of packet handling beyond basic forwarding and simple

header modifications. Of special interest is the ability to capture and embed instantaneous state from the network element and encode this state directly into the transit packet. One example would be to record the transit device's name, ingress and egress port identifiers, queue depths, timestamps and so on. By collecting this state along each network device in the path, it becomes trivial to trace a probe's path through the network as well as record transit device characteristics. Extending this model, one could build a tool that combines the useful properties of ping and traceroute using a single packet flight through the network, without the constraints of control plane (aka "slow path") processing. To aid in the development of such tooling, this document defines a format for embedding telemetry information in the body of active probing packets.

## 2. Data plane probe

This section defines the structure of the active data-plane probe packets.

### 2.1. Probe transport

This document assumes the use of IP/UDP for data-plane probing (either IPv4 or IPv6). A receiving application may listen on a pre-defined UDP port to collect and possibly echo back the information embedded in the probe. One potential limitation to this methodology is the size of the probe packet, as some data-plane faults may only impact packets of a given size or range of sizes. In this case, the data-plane probe may not be able to detect such issues, given the requirement to pre-allocate storage in the packet body.

### 2.2. Probe structure

The sender is responsible for constructing a packet large enough to hold all records to be added by the network elements. Concurrently, the probes must not exceed the minimum MTU allowed along the path, so it is assumed that the sender either knows the needed MTU or relies on well-known mechanisms for path MTU discovery. After adding the mandatory protocol (IP, UDP, etc.) headers, the packet payload is built according to the following layout:

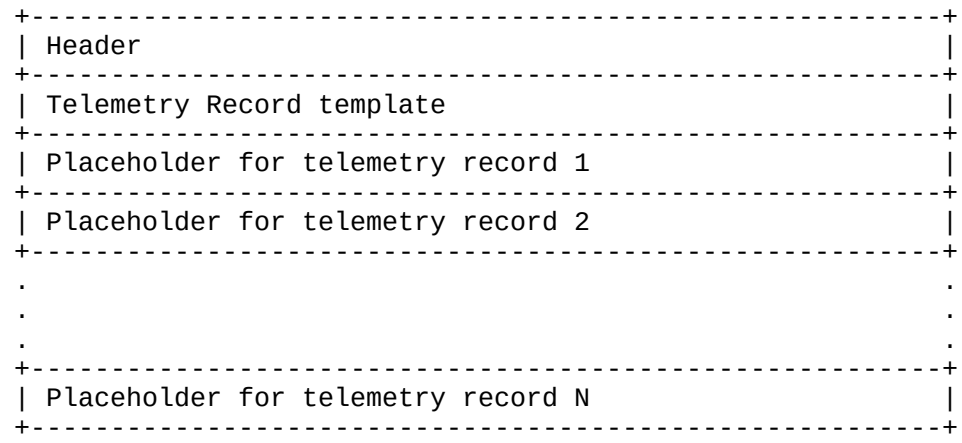



Figure 1: Probe layout

Notice that all record placeholders are equal size, as prescribed by the telemetry record template, and that space for those must be pre-allocated by the sender of the packet. Each record corresponds to a single network element on the path from sender to receiver of the packet.

## 2.3. Header Format

The probe payload starts with a fixed-size header. The header identifies the packet as a probe packet, and encodes basic information shared by all telemetry records.

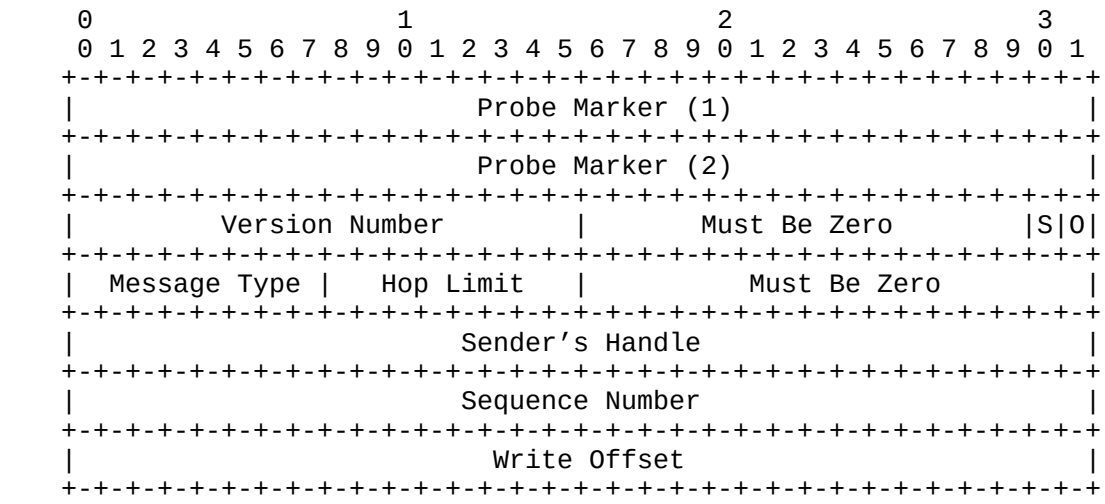



Figure 2: Header Format

(1) The "Probe Marker" fields are arbitrary 32-bit values generally used by the network elements to identify the packet as a probe packet. These fields should be interpreted as unsigned integer values, stored in network byte order. For example, a network element may be configured to recognize a UDP packet destined to port 31337 and having 0xDEAD 0xBEEF as the values in "Probe Marker" field as an active probe, and treat it respectively.

(2) "Version Number" is currently set to 1.

(3) The "Global Flags" field is 8 bits, and defines the following flags:

    (1) "Overflow" (O-bit) (least significant bit). This bit is set by the network element if there is no record placeholder available: i.e. the packet is already "full" of telemetry information.

    (2) "Sealed" (S-bit). This bit instructs the network element to forward the packet WITHOUT embedding telemetry data, even if it matches the probe identification rules. This mechanism could be used to send "realistic" probes of arbitrary size after the network path associated with the combination of source/destination IP addresses and ports has been previously established. The network element must not inspect the "Telemetry Record Template" field for "sealed" probes.

(4) The "Message Type" field value could be either "1" - "Probe" or "2" - "Probe Reply"

(5) "Hop Limit" is defined only for "Message Type" of "1" ("Probe"). For "Probe Reply" the "Hop Limit" field must be set to zero. This field is treated as an integer value and decremented by every network element in the path as "Probe" propagates. See the Section 4 section on the intended use of the field.

(6) The "Sender's Handle" field is set by the sender to allow the receiver to identify a particular originator of probe packets. Along with "Sequence Number" it allows for tracking of packet order and loss within the network.

(7) The "Write Offset" field specifies the offset for the next telemetry record to be written in the probe packet body. It counts from the start of the packet body and must be initially set to the first octet after the "Record Template" field. It must be incremented by every network element that adds a telemetry record, without overflowing the storage. This simplifies the work for the subsequent network element - it just needs to parse the template and then add the data at the "Write Offset".

## 2.4. Telemetry Record Template

The following figure defines the "Record Template". This template uses type-length fields to describe the telemetry data records as added by network elements. The most significant bit in the "Type" field must be set to zero.

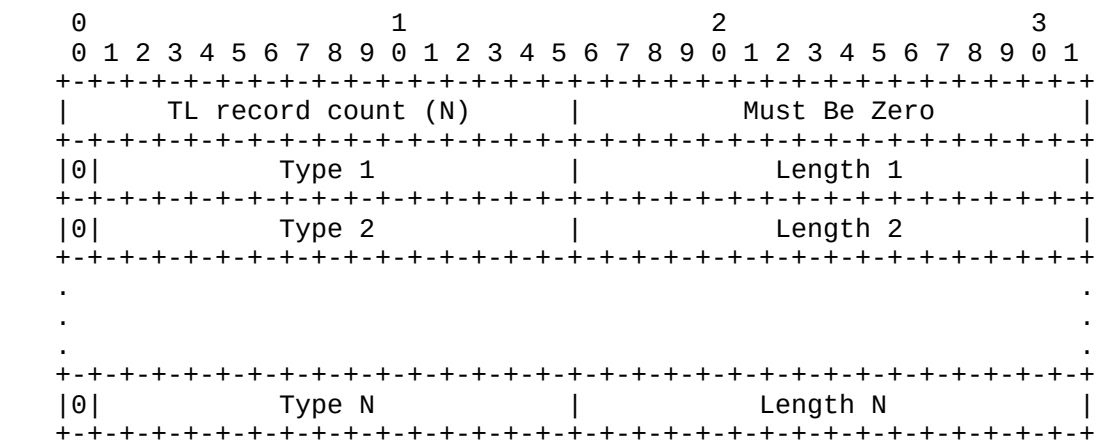



Figure 3: Record Template

### 2.5. Telemetry Record

This section defines the structure of a telemetry record. Every network element capable of reporting inband telemetry data must add a record as defined in the "Record Template" to the probe packet. The new record must be inserted at the "Write Offset" position in the packet payload, with the "Write Offset" subsequenly incremented by the size of the new record. The order of TLV elements must follow the order prescribed by the Figure 3 portion of the probe packet. The most significant bit in the type field ("S-bit") must be set to "1" if the network element was able to understand and record the requested telemetry type. That bit must be set to zero otherwise, along with the contents of the "Value" field. The length field is the TLV field length including the "Type" and "Length" fields.

If writing a new telemetry record to the packet body would cause it to exceed the packet size, no record is added and the overflow "O-bit" must be set to "1" in the probe header.

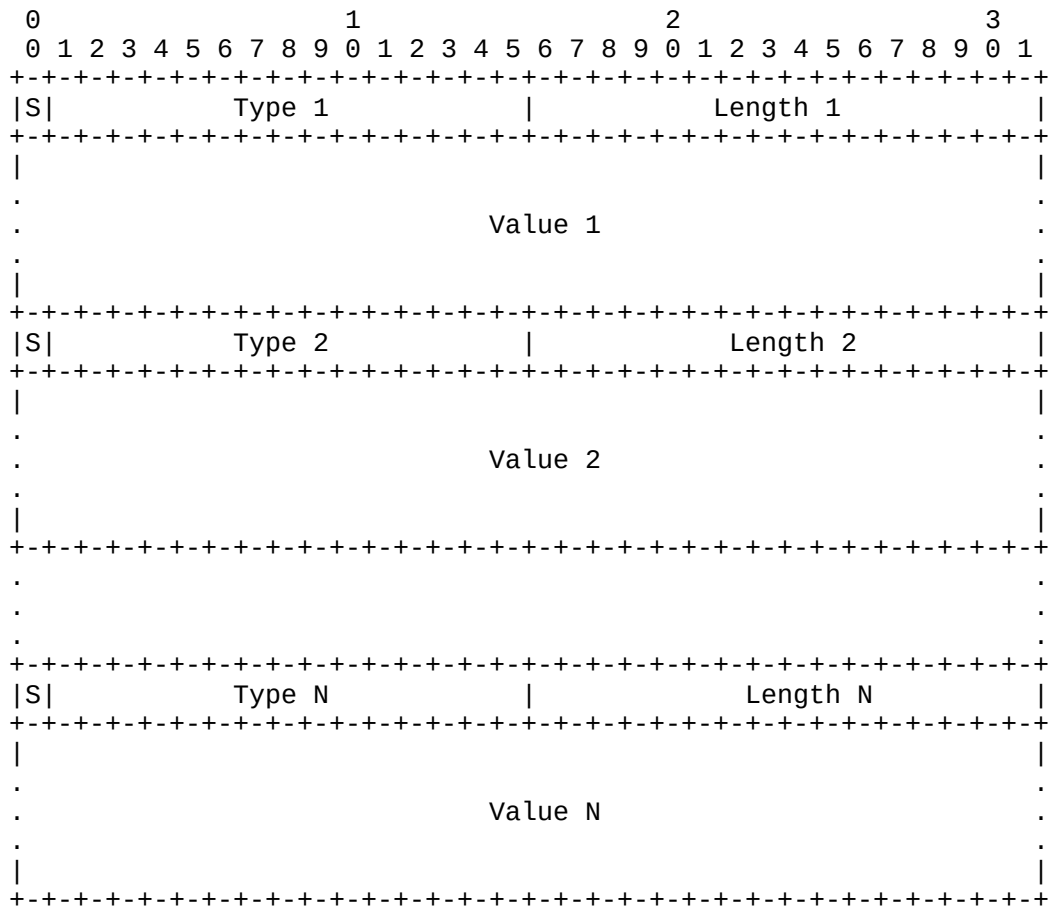



Figure 4: Telemetry Record Format

## 3. Telemetry Record Types

This section defines some of the telemetry record types that could be supported by the network elements.

### 3.1. Device Identifier

This is used to identify the device reporting telemetry information. This document does not prescribe any specific identifier format.

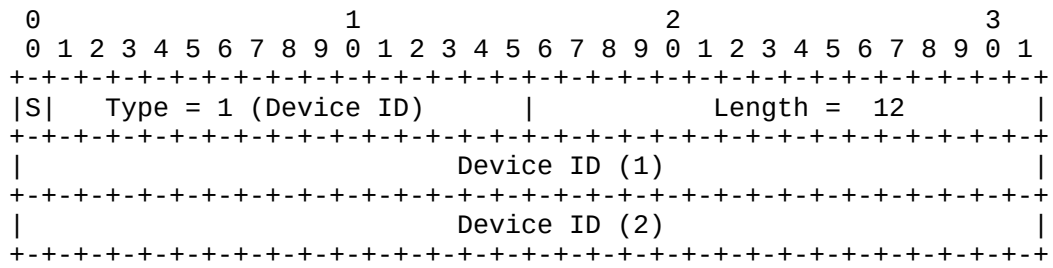



Figure 5: Device Identifier

## 3.2. Timestamp

This telemetry record encodes the time that the packet enters and leaves the device, in UTC. The "entering" time is recorded when the L2 header enters the processing pipeline. The "exit" time is recorded when the network elements starts serializing L2 header on egress port.

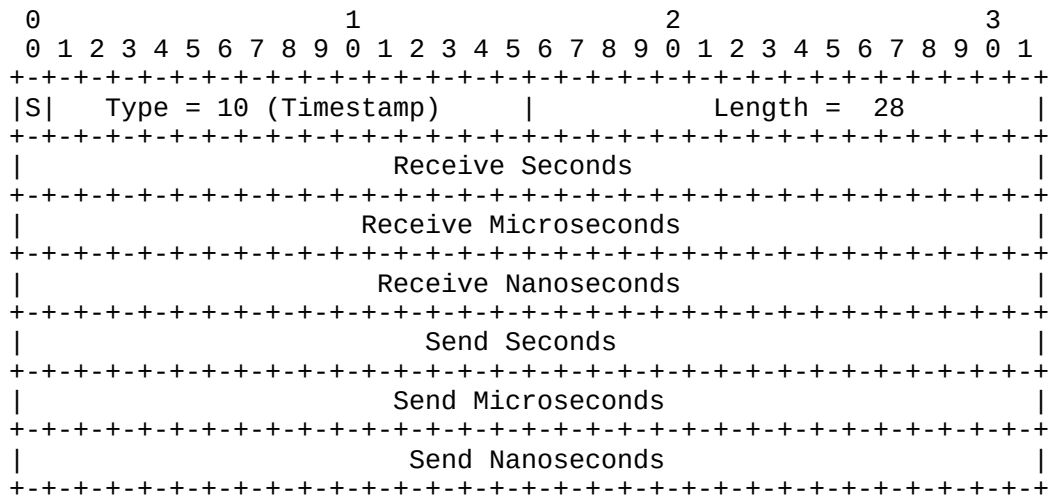



Figure 6: Timestamp

## 3.3. Queueing Delay

Encodes the amount of time that the frame has spent queued in the network element. This is only recorded if packet has been queued, and defines the time spent in memory buffers. This could be helpful to detect queueing-related delays in the network. In case of the cut-through switching operation this must be set to zero.

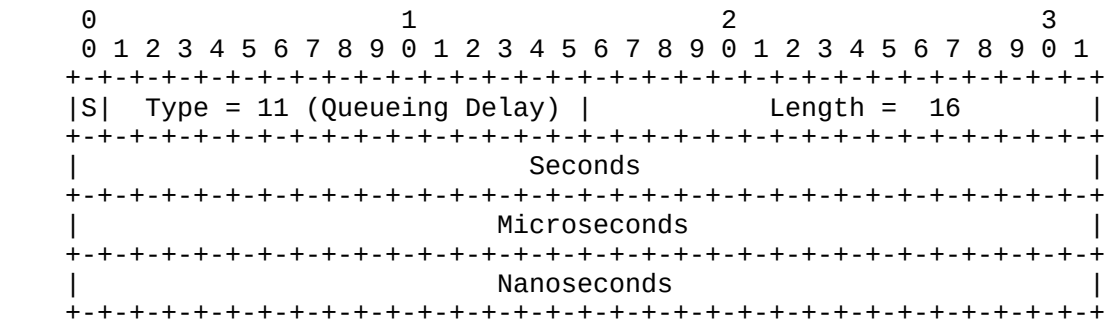



Figure 7: Queueing Delay

## 3.4. Ingress/Egress Port IDs

This record stores the ingress and egress physical ports used to receive and send packet respectively. Here, "physical port" means a unit with actual MAC and PHY devices associated - not any logical subdivision based, for example, on protocol level tags (e.g. VLAN). The port identifiers are opaque, and defined as 32-bit entries.

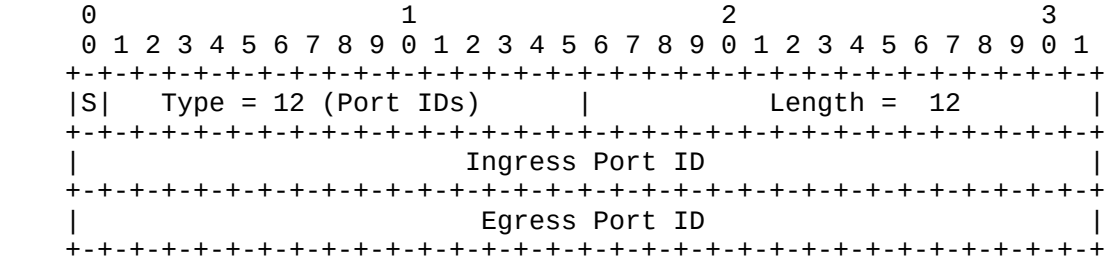



Figure 8: Ingress/Egress Port IDs

## 3.5. Forwarding Information

Records defined in this section require the network element to store forwarding information that was used to direct the packet to the next-hop. In the network that uses multiple forwarding plane implementations (e.g. IP and MPLS) the originator of the probe is required to populate the record template with all kinds of forwarding information it expects in the path. The network elements then populate the entries they know about, e.g. in IPv4-only network the "IPv6 Route" record will be left unfilled, and so will be "MPLS Route".

### 3.5.1. IPv6 Route

This record stores the IPv6 route that has been used for packet forwarding. If not used, then S-bit is set to zero, along with the value field.

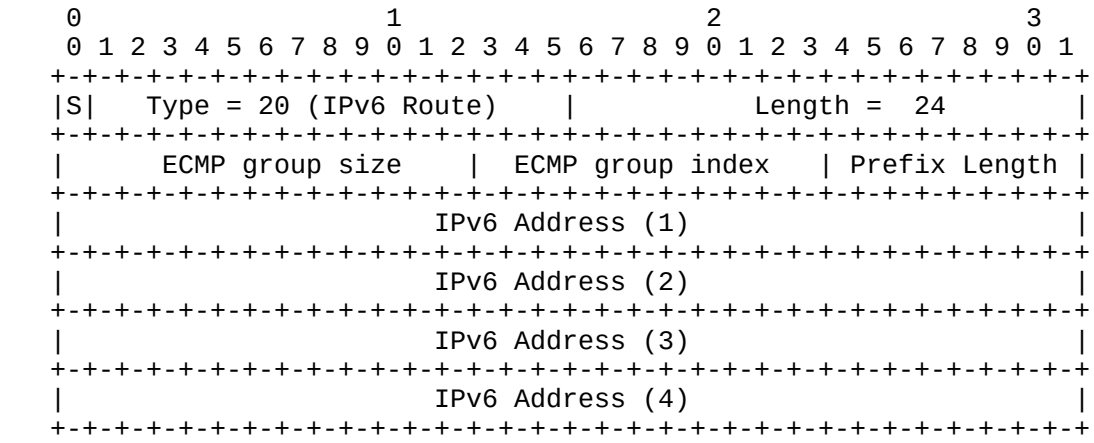



Figure 9: IPv6 Route

### 3.5.2. IPv4 Route

This record stores the IPv4 route that has been used for packet forwarding. If not used, then S-bit is set to zero, along with the value field.

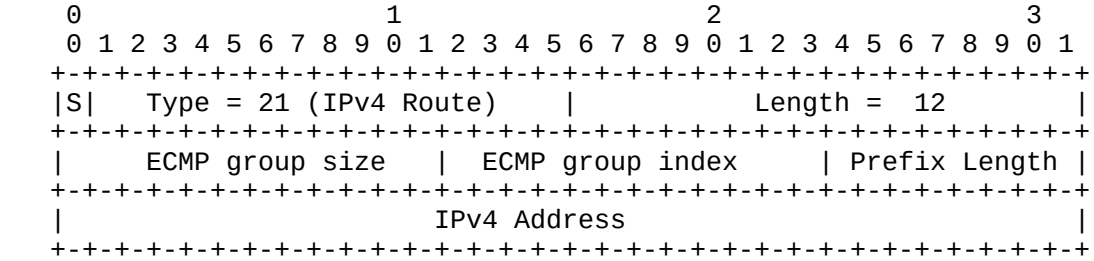



Figure 10: IPv4 Route

### 3.5.3. MPLS Route

This record stores the MPLS label mapping that has been used for packet forwarding. It is possible that inbound or outbound label set set to zero, if it was not used (e.g. on ingress or egress of the domain). At the edge of IP2MPLS or MPLS2IP domain it is expected

that the device would fill in the "MPLS Route" telemetry record along with the corresponding "IPv6 Route" or "IPv4 Route" records.

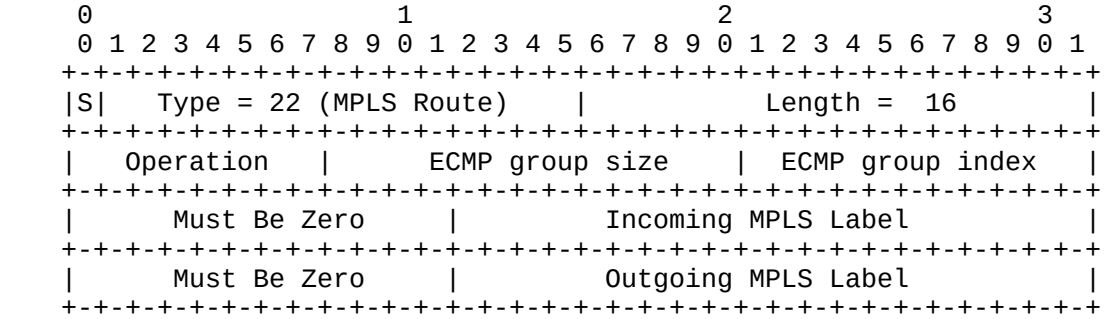



Figure 11: MPLS Route

There are three MPLS operations defined

"1" - Push

"2" - Pop

"3" - Swap

## 4. Operating in loopback mode

In "loopback" mode the flow of probes is "turned back" at a given network element. The network element that "turns" packets around is identified using the "Hop Limit" field. The network element that receives a "Probe" type packet having "Hop Limit" value of "1" is required to perform the following:

Change the "Message Type" field to "Probe Reply" and set the "Hop Limit" to zero.

Swap the destination/source addresses and port values in the IP/UDP headers of the probe packet.

Add a telemetry record as required using the newly build IP/UDP headers to determine forwarding information.

This way, the original probe is routed back to originator. Notice that the return path may be different from the path that the original probe has taken. This path will be recorded by the network elements as the reply is transported back to the sender. Using this technique one may progressively test a path until its breaking point. Unlike

the traditional traceroute utility, however, the returning packets are the original probes, not the ICMP messages.

## 5. Processing Probe Packet

### 5.1. Detecting a probe

Since the probe looks like a regular UDP packet, the data-plane hardware needs a way to recognize it for special processing. This document does not prescribe a specific way to do that. For example, classification could be based on only the destination UDP port, or using more complex pattern matching techniques, e.g matching on the contents of "Probe Marker" field.

## 6. Non-Capable Devices

Non-capable devices are those that cannot process a probe natively in the fast-path data plane. Further, there could be two types of such devices: those that can still process it via the control-plane software, and those that can not. The control-plane processing should be triggered by use of the "Router-Alert" option for IPv4 of IPv6 packets (see [RFC2113] or [RFC2711]) added by the originator of the probe. A control-plane capable device is expected to interpret and fill-in as much telemetry-record data as it possibly could, given the limited abilities.

Network elements that are not capable of processing the data-plane probes are expected to perform regular packet forwarding. If a network element receives a packet with the router-alert option set, but has no special configuration to detect such probes, it should process it according to [RFC6398]. Absence of the router alert option leaves the non dataplane-capable devices with the only option of processing the probe using traditional forwarding.

## 7. Handling data-plane probes in the MPLS domain

In general, the payload of an MPLS packet is opaque to the network element. However, in many cases the network element still performs a lookup beyond the MPLS label stack, e.g. to obtain information such as L4 ports for load balancing. It may be possible to perform data-plane probe classification in the same manner, additionally using the "Probe Marker" to distinguish the probe packets.

In accordance to [RFC6178] Label Edge Routers (LERs) are required not to impose an MPLS router-alert label for packets carrying the router-alert option. It may be beneficial to enable such translation, so that an end-to-end validation could be performed if a control-plane capable MPLS network element is present on the probe’s path.

## 8. Multi-chip device considerations

TBD

## 9. IANA Considerations

None

## 10. Acknowledgements

The author would like to thank L.J. Wobker and Changhoom Kim for reviewing and providing valuable comments for the initial version of this document.

## 11. References

### 11.1. Normative References

[RFC2113] Katz, D., "IP Router Alert Option", RFC 2113, DOI 10.17487/RFC2113, February 1997, <http://www.rfc-editor.org/info/rfc2113>.

[RFC2711] Partridge, C. and A. Jackson, "IPv6 Router Alert Option", RFC 2711, DOI 10.17487/RFC2711, October 1999, <http://www.rfc-editor.org/info/rfc2711>.

[RFC6398] Le Faucheur, F., Ed., "IP Router Alert Considerations and Usage", BCP 168, RFC 6398, DOI 10.17487/RFC6398, October 2011, <http://www.rfc-editor.org/info/rfc6398>.

[RFC6178] Smith, D., Mullooly, J., Jaeger, W., and T. Scholl, "Label Edge Router Forwarding of IPv4 Option Packets", RFC 6178, DOI 10.17487/RFC6178, March 2011, <http://www.rfc-editor.org/info/rfc6178>.

### 11.2. Informative References

[RFC7276] Mizrahi, T., Sprecher, N., Bellagamba, E., and Y. Weingarten, "An Overview of Operations, Administration, and Maintenance (OAM) Tools", RFC 7276, DOI 10.17487/RFC7276, June 2014, <http://www.rfc-editor.org/info/rfc7276>.

## Author’s Address

Petr Lapukhov
Facebook
1 Hacker Way
Menlo Park, CA 94025
US

Email: petr@fb.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

P. Lapukhov
Facebook

October 31, 2016

# Deploying Identifier-Locator Addressing (ILA) in datacenter networks
## draft-lapukhov-ila-deployment-01

Abstract

Identifier-Locator Addressing architecture defined in [I-D.herbert-nvo3-ila] proposes the use of locator-identifier split in IPv6 address to realize workload mobility and more efficient use of network resources. This document describes how ILA can be implemented in datacenter using BGP as the control-plane protocol. Generally speaking, ILA could be built using different control planes, and BGP is one particular instantiation. The motivation is BGP being a well-known protocol, sufficient for small to medium size deployments, on scale of few millions of identifier to locator mappings. Defining more generic and scalable control plane variants is outside of scope of this document.

Table of Contents

## 1. Introduction

This document provides high-level guidelines for building an ILA-enabled datacenter using BGP [RFC4271] as the protocol for ILA mapping information dissemination. The reader is expected to be familiar with the principles presented in [I-D.herbert-nvo3-ila]. Reading on ILNP architecture defined in [RFC6740] is also recommended, but not needed for understanding of this document. While ILA does not implement the original ILNP proposal, it's based on the same idea of maintaining the Identifier vs Locator split in the IPv6 address.

ILA benefits from routed datacenter networks, i.e. networks that do not rely on spanning Layer-2 domains across multiple network devices. Endpoint mobility made possible by ILA is one of the key benefits ILA brings to the datacenter networks. Combining ILA with fully routed network design allows for achieving the robustness of routed network with the flexibility of endpoint mobility. Some practical recommendations for building a fully-routed datacenter network could be found in [RFC7938] or [ROUTED-DESIGN].

Though workload mobility could also be achieved in L3 switched networks by using "host-route injection" technique, such approach has limited applicability, due to high stress put on the underlying control and data planes. The mobile prefix needs to be removed, re-injected and propagated to all network devices every time an address moves.

ILA is an alternative to "encapsulation" approaches, such as LISP ([RFC6830]), for realizing the endpoint mobility and network virtualization. Using simple address rewrites significantly reduces the processing overhead on the hosts, and makes various hardware and software network acceleration functions easier to implement (e.g. checksum computation offload). Furthermore, ILA keeps the underlying network fully visible to the applications that use ILA addresses, which makes network troubleshooting easier, as compared to the "encapsulation" approaches.

## 2. Terminology

This section defines ILA-specific terminology that will be used through the document.

ILA domain: a collection of ILA hosts and ILA routers that collectively support ILA identifier mobility and network virtualization model. The ILA domain is assigned a single 64-bit IPv6 prefix known as SIR (Standard Identifier Representation, see [I-D.herbert-nvo3-ila]) prefix, which is made known to all hosts and routers in the domain. This prefix is used to construct the complete 128-bit IPv6 addresses for ILA identifies found in the domain.

SIR Address: IPv6 address constructed from SIR prefix concatenated with the 64-bit identifier. This is the address visible to the applications and transport layer on ILA hosts.

ILA Address: IPv6 address constructed from actual valid 64-bit locator and 64-bit identifier. This address is what being seen by transit network devices - it is expected to be routable in the underlying network.

ILA mapping table: The table for mapping identifiers to locators present in ILA host or ILA router. This table is updated either via BGP, or ILA redirect messages. ILA routers maintain full authoritative copy of the table, while ILA hosts may have their own smaller view of the global mapping state.

ILA host: network endpoint that is capable of accepting and originating packets with ILA addresses, by performing stateless rewrite between SIR addresses and ILA addresses. The host maintains its own local version of the ILA mapping table and has at least one ILA locator (64-bit prefix) assigned.

Non-ILA host: network endpoint that is not aware of ILA addressing structure and does not participate in ILA address translations. To this host, the SIR and ILA addresses look like regular IPv6 addresses.

ILA router: network endpoint that is responsible for two main functions:

* Storing and disseminating the authoritative ILA mapping information within the ILA domain (NVA role per [I-D.ietf-nvo3-arch]).

* Serving as the gateway between the ILA-hosts and non-ILA hosts, as well as the gateway for communicating with other ILA domains (NVE role per [I-D.ietf-nvo3-arch]).

Task: the unit of mobility in ILA domain. Each task is assigned an identifier unique within the ILA domain, which follows the task

as it changes the hosts and, consequently, the locators. Implementation wise, the task can run within a container or a virtual machine, for example.

Tenant: owner of the tasks executed in the shared environment.

Common Locator Address (CLA): Special ILA address constructed as <locator>::1 and identifying the physical host itself. This address is used to send and receive of the ILA redirect messages.

## 3. ILA deployment process

The ILA domain consists of the following conceptual elements:

- Routed network that provides reachability among physical hosts, i.e. provides routing within the locator address space.
- ILA hosts, each assigned a unique /64 prefix reachable within the network. ILA hosts maintain their own local version of ILA mapping table.
- ILA routers, each injecting the domain's SIR prefix into the routed network and maintaining the full mapping table for the ILA domain. The routers could be implemented in software, or using specialized hardware appliances.
- Centralized BGP router-reflector nodes that peer with all of the ILA hosts and all of the ILA routers within the domain for the purpose of mapping information dissemination. ILA hosts and routers run the BGP processes to communicate with the reflectors.

Deploying ILA in datacenter requires the following logical steps:

- Preparing the network. Assigning locator addressing to the hosts (servers) in the network and providing routed interconnection among the locator prefixes.
- Configuring ILA hosts and ILA routers. Each ILA domain requires a set of ILA routers to facilitate mapping function and provide connectivity to other ILA domains and the Internet. Each ILA domain is assigned a /64 SIR prefix, which scopes all identifiers in the domain. All ILA hosts and ILA routers within a domain are aware of the SIR prefix of this domain.
- Enabling the ILA control plane. Configuring the BGP mesh for mapping information dissemination within the ILA domain and injecting the SIR prefix into routed network from the ILA routers to facilitate communications among the ILA domain and from / to

the Internet. See [I-D.lapukhov-bgp-ila-afi] for definition of the corresponding BGP extension.

- o Deploying an address management solution to coordinate allocation of ILA identifiers. In simplest cases, the addresses could be generated on each host individually, without central coordination.

## 4. Preparing the network

This section provides overview of the network-related configuration needed for ILA.

### 4.1. Data-center network topology

For ease of reference, this document adopts the Clos topology described in [RFC7938] along with the terminology developed in that document.

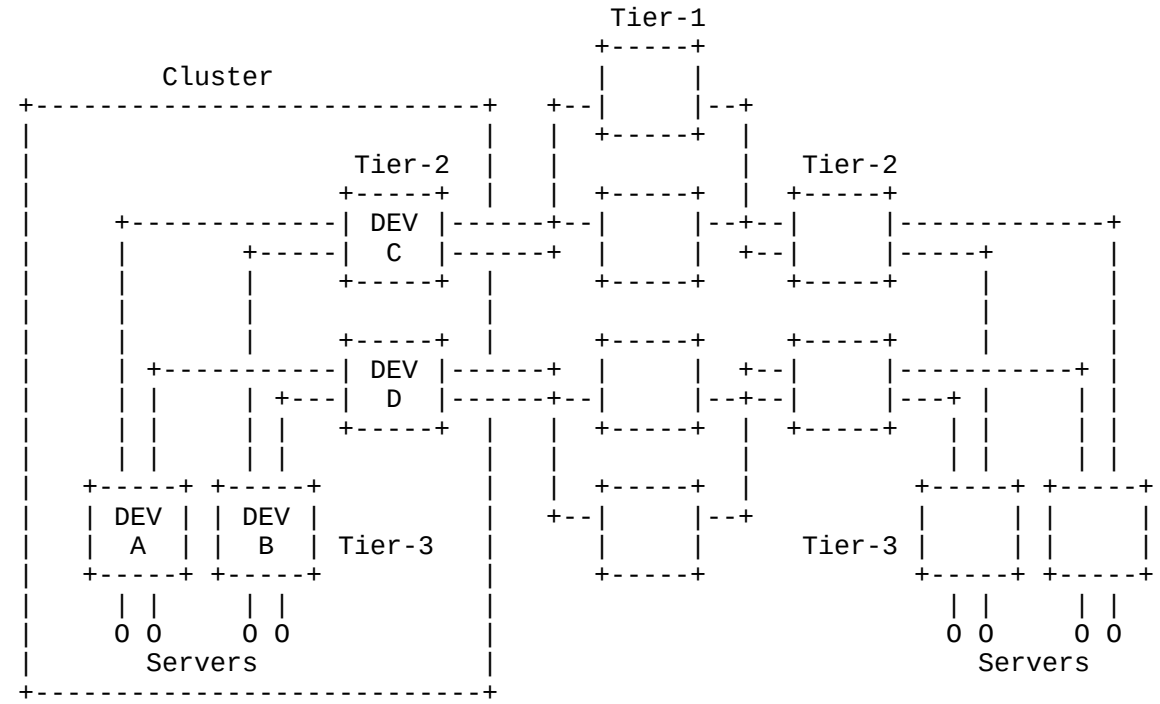



Figure 1: 5-Stage Clos topology

The network is partitioned hierarchically in three tiers, with tier numbering starting at the "middle" stage of the Clos network. The "middle" tier is often called as the "spine" of the network.

A set of directly connected Tier-2 and Tier-3 devices along with their attached servers will be referred to as a "cluster".

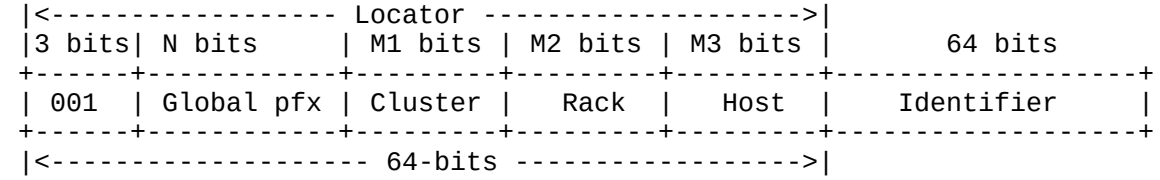
Tier-3 switches that connect the servers, are often referred to as "ToR" (Top of Rack) switches or simply "rack switches".

## 4.2. Configuring locator addressing

A mandatory prerequisite for ILA deployment is enabling IPv6 routing in the network. This could be done using either dual-stack IPv4/IPv6 deployment or IPv6-only deployments. This document assumes the network has been already configured to forward IPv6 traffic. See [I-D.ietf-v6ops-dc-ipv6] for operational considerations on deploying IPv6 in the datacenter.

ILA requires every ILA host to have at least one 64-bit locator assigned. This means that every host (server) in the datacenter network needs to have at least one /64 IPv6 prefix configured on one of its interfaces. These /64 prefixes could be either globally routable or unique-local.

The use of the globally routable addressing scheme allows for deploying highly scalable hierarchical addressing scheme, and make the locators accessible from the Internet. The figure below illustrates the structure of a globally-routable locator:



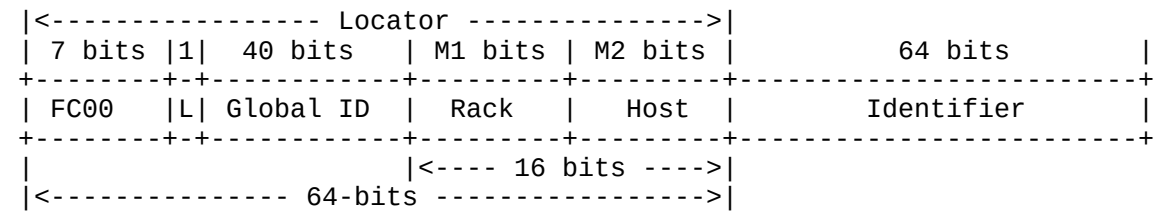

For example, a global /32 prefix (N=29) allows for sub-allocation of 2^32 locators. This sub-allocation could be done hierarchically, mapping to the tiers of network topology. Following the /32 example prefix:

Allocate 256 /64 prefixes per Tier-3 switch (M3 = 8 bits), which allows for up to 256 physical hosts in a rack, with /56 prefix assigned per rack.

Assuming 256 Tier-3 switches per cluster, one would allocate /48 per cluster (M2 = 8 bits).

This leaves room for 16-bits (64K) cluster per datacenter (M1 = 16 bits). This space could be further sub-divided if multiple Clos network fabrics have been deployed.

The use of unique-local addressing for locators is more limiting in terms of available space, as it only offers 16-bits for sub-allocation. It does, however, have the benefit of ad-hoc allocation. This could work better for smaller deployment, e.g. allocating 10-bits to enumerate Tier-3 switches (physical racks of servers) and 6 bits to enumerate hosts within a rack. For instance, the address structure may look as following, here M1 = 10 bits and M2 = 6 bits.




In either case, the addressing scheme is hierarchical, allowing for simple route summarization logic and better routing system scaling (see [RFC2791]). This is especially important in case of IPv6, since contemporary datacenter network switches often have smaller IPv6 lookup tables as compared to IPv4. Route summarization also requires certain network design changes to avoid packet black-holing under link failures. This problem gets more complicated in Clos topologies, and analyzed in more details in [RFC7938].

In greenfield deployments, each ILA host could be assigned a /64 locator prefix prefix during provisioning phase. There are multiple options to accomplish this:

- Assigning static link-local addresses to servers and statically routing /64 prefixes from Tier-3 switches to the servers over those link-local addresses. In this model, the operator would plan and pre-allocate per ILA-host prefixes beforehand, and configure the Tier-3 switches accordingly. From operational risks perspective, if the server is not present while the static route is configured on Tier-3 switch, packets destined to the corresponding /64 prefix will cause the switch to continuously generate IPv6 NDP packets ("gleaning"), which puts extra stress on the device’s CPU.

- The servers may request the /64 prefix using IPv6 Prefix Delegation mechanism as defined in [RFC3633]. This allocation

could be made "permanent" by proper DHCPv6 server configuration and ensuring the same prefix is always being delegated to the same server. The Tier-3 switch would act as DHCPv6 relay and will install the corresponding /64 IPv6 route dynamically. This approach addresses both the allocation and the routing problem, but makes the setup potentially more fragile operationally (reliance on additional protocol) and harder to debug (additional process involved).

- The server may run a routing daemon (e.g. BGP process) and inject the pre-allocated /64 prefix into Tier-3 switch. The address allocation in this case needs to happen by some other means. This is more suitable for ad-hoc ILA testing and small, rapid deployments.

The server itself may use one of the IPv6 addresses in /64 prefix for its own addressing, e.g. for remote access or management purposes. Alternatively, the server may obtain another IPv6 address from a different (non-locator) IPv6 address range allocated for the datacenter. This document recommends using <locator>::1 as the special identifier, naming it as "Common Locator Address" (CLA). Such choice of identifier make it easy to differentiate from regular identifiers. This identifier could be used for connectivity testing.

Route summarization for the locator prefixes is highly desirable to reduce the stress on the network switches forwarding tables and improve control-plane stability, and need to be implemented at least on Tier-3 switches. In simplest case, the switches could be statically preconfigured with the summary routes. These routes need to agree with the prefixes that are assigned to the servers, especially in the case when dynamic prefix injection is used. As a possible alternative, simple virtual aggregation could be employed, where hosts inject both the specific and the summary route, and installation of corresponding FIB entries is suppressed as per the rules defined in [RFC6769]. The latter approach does not improve the control plane scalability, but solves the issues with packet black-holing in presence of network summarization. It also requires the network hardware support, which may not be present.

In retrofitting scenarios, the servers are likely to already have 128-bit IPv6 addresses assigned, allocated from the datacenter address space, e.g. by using a single /64 prefix per Tier-3 switch. In this case, the additional locator prefix needs to be assigned in the same way as described above for greenfield deployments. The only difference is that the new prefix and the old server address may be allocated from different IPv6 address ranges.

## 5. Deploying ILA routers

ILA routers perform multiple functions within the ILA domain:

- Serve as the centralized store of the identifier-to-mapper information in the domain. The mappings are delivered to the ILA routers as described in Section 7.

- Act as the gateway between the ILA hosts and non-ILA capable hosts, e.g. the Internet.

The ILA hosts will send the packets destined to identifiers they don’t have mappings for to the ILA routers initially to perform the ILA translation, and the hosts outside of the ILA domain will use the ILA routers for all communications with the domain. The ILA routers may also act as ILA hosts and have one or more identifiers assigned.

### 5.1. ILA Redirect Message

ILA routers may originate and ILA hosts must receive and process ILA redirect messages. The ILA redirect message is carried in UDP packet and destined toward a well-known port. It carries the information binding an identifier to its locator. For security purposes, this message is expected to be authenticated by cryptographic means, such as by using keyed HMAC (message authentication code) procedure. Every host in the domain is then required to be configured with the key information to be able to validate the redirecte messages.

The ILA redirect message might be signed with multiple HMAC keys to facilitate key transition in the domain. The redirect message will carry multiple signatures along with corresponding numeric key-identigier, and the ILA hosts are expected to use the signature with the highest locally known identifier. As the old key leaves rotation, eventually every host will get updated and the signature made using the old key could be removed.

### 5.2. Configuration parameters

The ILA routers need the following configured for their operation:

- Regular, non-anycast 128-bit IPv6 address to connect the ILA router to the datacenter network.

- Cryptographic material to authenticate ILA redirect messages, for example key to be used with HMAC scheme.

- The /64 SIR prefix for the ILA domain, shared by all ILA routers. This prefix is advertised into the network in anycast fashion and

"intercepts" all traffic destined from hosts outside of ILA domains to the SIR addresses in the domain. The prefix could be injected in "always-on" fashion, e.g. by using BGP injectors on ILA routers. This couples the ILA router's life-cycle with the prefix injection cycle.

- Control-plane configuration, i.e. the IPv6 addresses of BGP route reflectors, and possibly some configuration for the local BGP process. This is discussed in more details in Section 7.

- Management settings, such as maximum rate of ILA redirect messages, and associated security attributes (e.g. the key pair used for message signing).

- A configuration flag that instructs the router whether the ILA redirect messages needs to be sent out. The ILA router does not receive ILA redirect messages, since by design it knows of all active mappings in the domain.

### 5.3. ILA router operation

Upon booting, the ILA router is first required to join the control plane mesh and learn of the mappings that exist in the ILA domain. It is also aware of the SIR prefix that is used within its domain. After the router has learned of the mappings, it may inject the anycast SIR prefix in the datacenter network and join the operational group of ILA routers.

Just like any ILA node, the ILA router is required to have a 64-bit locator configured. Special identifier ::1 is used to build the source and destination addresses of the ILA redirect messages.

When ILA router receives a packet with the upper 64-bits of the destination IPv6 address matching its configured SIR prefix, it performs the following:

- If the destination address does not match the SIR prefix, the ILA router discards the packet, as it is not supposed to be received by the ILA router.

- Attempts to resolve the source identifier (bottom 64-bits of the source address), if applicable. If the source address matches SIR prefix, it is coming from an ILA host. The route then needs to translate the identifier found in the source address to its locator. If the translation fails, send back the ILA "Mapping Not Found" message. If the source address does not match the SIR prefix, then no translation is needed, and no redirect messages need to be sent back.

- o Attempts to find the locator matching for the destination identifier (the bottom 64-bits of the destination IPv6 address). If the mapping for destination identifier is not found, the original packet is dropped, and an ICMPv6 "Destination Unreachable" message, type "3" is sent back to the message originator. Otherwise, the router does the following:

  - * Rewrites the SIR prefix in the destination IPv6 address with the new locator and forwards the packet back to the network.

  - * If sending of ILA redirect messages is permitted, the router sends the ILA redirect message back to the originator of the packet, by looking up the source identifier and finding the corresponding locator. The redirect informs the source of the actual destination locator. The redirect messages must be rate-limited to avoid sending ILA redirect for every incoming IPv6 packet.

  - * As mentioned previously, the source and the destination ILA addresses of the redirect message IPv6 header use the identifier value "::1", which designted them to be develired to the ILA control process.

If the source IPv6 address check reveals that the packet is not coming from the ILA domain the router belongs to (i.e. the SIR prefix does not match), the ILA router does not need to send back the ILA redirect message, but instead simply continue to forward the packet as if the locator for the destination identifier could be found. The ILA router will still send the ICMPv6 "Destinationa Unreachable" message for unknown mappings.

### 5.4. Scaling considerations

Due to high load and reliability concerns, the ILA domain needs multiple ILA routers. The simplest way to provide redundancy is by letting the ILA routers inject the /64 SIR IPv6 prefix into the datacenter network in anycast fashion ([RFC4786]). This will allow to naturally use the datacenter network's Equal-Cost Multipath (ECMP) capabilities to distribute traffic among the ILA routers.

For redundancy purposes, the ILA routers would need to be spread across multiple physical racks in the datacenter. More ILA routers could be added incrementally to reduce the load and scale capacity horizontally, and join the operational ILA group in non-disruptive fashion, after they have learned the full mapping table for the ILA domain.

Use of anycast method does have some resulting routing implications. For example, using the network described in Section 4.1 will result in ILA hosts preferring to use the ILA routers in the same cluster, since those are closer based on the routing metric. Thus, the network may not evenly spread their packets across all ILA routers in the datacenter. It is therefore possible that some ILA routers will receive more traffic than the others. This issue is specific to anycast routing in general, and not specifically to ILA.

# 6. Deploying ILA hosts

This section reviews the deployment considerations for the ILA hosts.

## 6.1. Configuration parameters

The ILA hosts need to be configured with the following:

- SIR prefix of the ILA domain.
- IPv6 addresses of the BGP route reflectors.
- The routable /64 locator assigned to the host.
- ILA mapping entries expiration time, to time out unused entries.
- Cryptographic material to allow validation of redirect messages.
- Boolean flag, defining whether ILA redirection messages sending / receiving is enabled.

By disabling both the ILA mapping expiration time and the sending of ILA redirect messages the host is effectively configured for the "push" ILA mapping distribution distribution mode (see Section 8). In this mode, the BGP (control plane) is assumed to update/ synchronize all of the ILA mapping entries in response to the identifier move events, and redirect messages are not used.

The host is expected to recevive ILA redirect messages destined to its locator and identifier value of "::1". The source of such message must also use the identifier value of "::1" to be considered a redirect message.

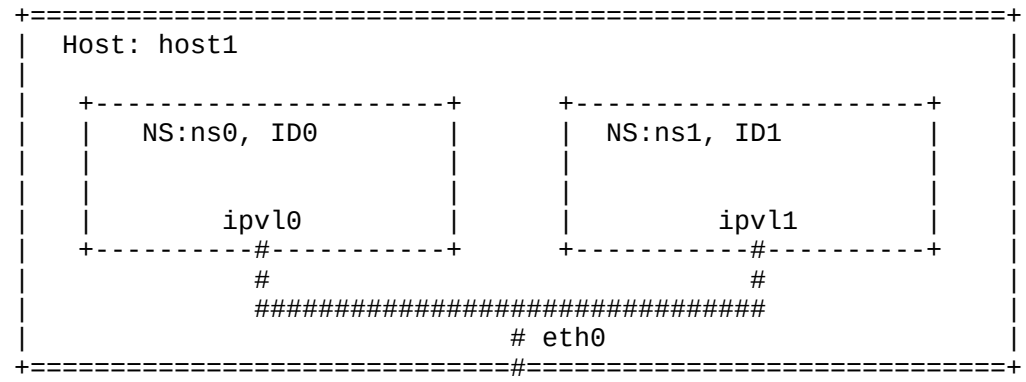
## 6.2. Providing task isolation

In simplest case, the host only needs to implement the ILA address rewrite function and inform the tasks starting on the host of the ILA addresses they can use. However, it might be desirable to provide the tasks with strong networking isolation guarantees, i.e. making

sure tasks are only allowed to use the IPv6 ILA address they have been allocated. For instance, with Linux operating system, this is possible by using the [LINUX-NAMESPACES] and [IPVLAN] techniques together.

Each task running on the host will be contained to its own networking namespace, and has the allocated ILA address bound to an interface that belongs to this namespace. The task would then only be able to bind to the single IPv6 ILA addresses delegated to the namespace.

With "ipvlan" technique, the packets arriving on physical host's NIC need to have their locator field adjusted before delivering to the task (the locator field is set to the /64 prefix assigned to the host). No additional routing lookups need to be performed on the physical host. On the egress path, all IPv6 lookups and rewrites happen in the default namespace, in Linux terminology. The figure below demonstrates a host with two tasks running, each in its own networking namespace. The namespace names are "ns0" and "ns1", and the corresponding task ILA identifiers are ID0 and ID1.




Tasks running in Linux namespaces with ipvlan

The use of "ipvlan"-like techniques is not strictly necessary. An alternative would be use the ILA host as a proper IPv6 router and treating the attached namespaces as hosts. This, however, has higher performance overhead, due to multiple forwarding lookups that need to be done in the kernel.

## 6.3. ILA host operation

When ILA host boots up, it joins the control-plane mesh by peering with the BGP route-reflectors. It may learn the active ILA mappings from the BGP route reflectors, or may initially keep the ILA mapping

table empty, depending whether "push" or "pull" distribution model has been selected.

When a tasks starts it will have an ILA identifier allocated, and the corresponding IPv6 address (built out of SIR prefix + the allocated identifier) bound to an interface within the networking namespace created for the task. The mapping is then propagated over BGP peering sessions to all ILA routers.

For outgoing packets, the ILA host performs the following:

- Matches the destination IPv6 address against the SIR prefix.
- If prefix matches, attempts to look-up the identifier portion of the address in the local ILA mapping table.
- If a match is found in ILA mapping table, rewrite the destination address and replace the SIR prefix with the actual locator.

For packets with destination IPv6 addresses that do not match the SIR prefix, usual forwarding rules apply. If no match is found for the SIR address, the packet is sent as is, and is expected to be delivered to the ILA routers, since those advertise the SIR prefix into the routing domain (without getting the locator portion rewritten - the packet has the SIR prefix in place of the locator).

For incoming packets, the ILA host should perform the following:

- Match their destination IPv6 addresses against the locator prefix (64 bits) of the host.
- If the destination address matches, deliver the packet to the corresponding namespace, based on the identifier portion.
- If the destination identifier in the incoming packet does not match any of the ILA mappings, and sending of ILA redirect message is enabled, the host sends an ILA redirect message back to the originator of the packet. The message will have an empty locator value, and informs the sender that the mapping it has for the identifier is no longer valid, prompting to erase the corresponding entry in the sender’s ILA mapping table.
- If the source address is SIR address, the receiving host may increase time-to-live for the corresponding mapping entry, if it is present in the ILA mapping table. This acts as a signal confirming liveness of the remote corresponding, and validity of the existing mapping. Otherwise, the mapping would be expired

based on the time-to-live provided by the original ILA redirect message, if ILA mapping expiration is enabled.

Sending an ILA redirect message by the ILA host requires the host to translate the source identifier of the original message. Assuming that flow was likely bi-directional, the entry should be readily available in the local ILA mapping table. If not, the ILA redirect message will be routed toward the originator via the ILA routers, i.e. sent back with locator equal to the SIR prefix. It is possible that both source and destination identifiers of the flow have moved, resulting in mutual sending of ILA redirect messages, and temporarily falling back to using the ILA routers.

If the ILA mapping entry expiration time is set to non-zero, the unused ILA mapping entries will eventually be deleted. The entry expiration needs to be disabled if the mappings are learned in event-driven fashion via the BGP mesh ("push" distribution mode).

## 7. Using BGP as the ILA control plane

This section discusses the use of BGP for ILA mapping information dissemination. The choice of BGP is made to allow for easier integration of hardware appliance, e.g. network switches with extended functionality, where BGP is commonly used as the control plane. Furthermore, BGP itself offers a simple way of disseminating data and converging on a key-value mapping across multiple nodes in eventually consistent fashion, and has proven track record of use in the industry. Furthermore, use of BGP allows for leveraging the monitoring extensions developed for the protocol. For example, [I-D.ietf-grow-bmp] could be used to observe ILA mapping changes in the network using existing tooling.

### 7.1. BGP topology

Per the common practice, a group of BGP route-reflectors (see [RFC4456]) should be deployed and peered over IBGP with all ILA hosts and ILA routers in the ILA domain. The reflectors themselves would also be peered in full-mesh fashion to provide backup paths for mapping information distribution, e.g. in case if one of reflectors loses a session to a host. Those reflectors do not need to be in the data-path, but merely serve for the purpose of information distribution. The number of route-reflectors should be at least two, to allow for redundancy. See below sections for discussion of route-reflection settings.

It is possible to co-locate the BGP route-reflectors with the ILA routers. This saves on having additional nodes for the purpose of just BGP route-reflection, but puts extra memory and CPU stress on

the ILA routers, and therefore is less desirable. Furthermore, it makes capacity-planning more difficult, and therefore is not recommended.

The route-reflectors are required to peer with potentially a very large number of ILA hosts, which may put scaling limits on the size of the ILA domain due to the overhead of maintaining large amount of BGP peering sessions. To alleviate this problem, the pool of ILA hosts may be split into "shards" and each shard would peer with a different group of route-reflectors. For example, the ILA domain may have four groups of route reflectors, each with four route-reflectors. The sixteen route-reflectors may then peer in a full-mesh fashion, to exchange the mappings they have received from the corresponding "shard" of the ILA domain. This method avoid the issues related to maintaining large amount of TCP sessions, but every BGP route-reflector is still required to maintain the full ILA mapping table.

In addition to ILA AFI/SAFI's, other AFI/SAFIs could be configured on BGP speakers, e.g. using [I-D.lapukhov-bgp-opaque-signaling] for opaque information dissemination in the ILA domain, e.g. to facilitate in distributed address allocation.

### 7.2. Any-to-any mapping distribution

In this mode, the ILA routers could act as IBGP route-reflectors [RFC4456] for all of the IBGP sessions they have, and relay the mapping information among the ILA hosts. This would allow the hosts to avoid initially sending packets to the ILA routers, at the expense of maintaining the ILA mapping table. Additionally, this allows for completely disabling the ILA redirect messages and using only the mapping information propagated by BGP.

### 7.3. Hub-and-spoke mapping distribution

Alternatively, BGP could be used to deliver the mappings from ILA hosts to ILA routers only. The hosts and the routers would establish IBGP peering sessions with the route-reflectors in hub-and-spoke fashion, with BGP reflectors being the hubs. The ILA router sessions will be configured as the "route-reflector clients" on the route-reflectors, while the ILA hosts sessions will be left as ordinary IBGP sessions. This will propagate all needed mappings to the ILA routers and allow them to properly redirect the hosts. The ILA hosts are responsible for withdrawing and announcing the mappings as they change.

## 8. Push vs pull mapping distribution modes

The default mode of operations in ILA is "pull" mode, where mappings are learned by the ILA hosts via ILA redirect messages. Effectively, the process of populating the ILA mapping table is reactive and driven by data-plane events. In some case, e.g. upon identifier move, this may result in short periods of packet loss, while the sender receives the ILA redirect message and falls back to forwarding via the ILA routers. Furthermore, the use of ILA redirect messages requires security configuration to avoid message spoofing and cache poisoning attacks.

An alternative to "pull" mapping distribution on the hosts, is "push" mode, where all ILA hosts receive exactly the same mapping information as the ILA routers. In fact, every ILA host may even operate as an ILA router. In this case, the ILA message sending could be disabled in the ILA domain altogether. The "push" mode allows for proactive creation of the ILA mappings, and avoiding the packet loss, provided that the new mapping reaches the sending host before the destination identifier has moved. The trade-off here is the overhead of maintaining full mapping set on all ILA hosts.

For simplicity, this document recommends that all ILA hosts in the domain operate either in "push" or "pull" modes. In "push" mode the ILA mapping entries expiration needs to be turned off, along with sending of ILA messages. If an ILA host receives a packet for the ILA address it cannot map to locally, it is expected to send an ILA redirect message. If sending the ILA messages is disabled, the host must at least send an ICMPv6 "Destination Unreachable" message with code "3" - "Address Unreachable" to aid in debugging of missing mapping message. Notice that the ILA routers always operate in "push" mode, i.e. they only learn of mappings via the control plane exchange.

## 9. ILA address management

The ILA control plane and redirect messages perform mapping information dissemination, but the identifier allocation needs to be done separately. The address management process also depends on whether there is some hierarchy desired in the ILA namespace, e.g. if allocating a prefix per-tenant is needed.

### 9.1. Decentralized address management

In simplest case, each ILA host may independently allocate unique identifier per task when it first starts, and the task will retain it for the duration of its lifetime (see Appendix A of [I-D.herbert-nvo3-ila]). The chances of collision are very low given

the 60-bit value of the identifier. The scheduler is responsible for starting and moving the task in the ILA domain. The tasks belonging to the same tenant may discover each other’s addresses by some out-of-band signaling mechanism, e.g. a key-value store such as ([MEMCACHED]) or [ETCD] or use BGP for the same purpose as described in [I-D.lapukhov-bgp-opaque-signaling]. For instance, the task may publish its own identifier, consisting of the tenant name and task name, mapped to the SIR address of the task.

Decentralized allocation is still possible even if the unit of address allocation is prefix, e.g. when multiple tenants are sharing the infrastructure, and unique VNID (see [I-D.herbert-nvo3-ila] for definition) is needed per tenant to build the 96-bit prefixes allocated to tenants from the /64 SIR prefix. Since the size of VNID space is rather small, generating random VNIDs becomes more prone to collision. In this case, decentralized address allocation schemes, such as one described in [RFC7695] could be used. These techniques require the ILA nodes to have some shared communication medium for nodes to "claim" the prefixes and avoid collisions. Once again, various distributed key-value stores could be used to accomplish this.

### 9.2. Centralized address management

In the case where high level of control is needed to allocate the addresses, e.g. per-tenant prefixes, centralized address management schemes could be used in the ILA domain. This could be either proprietary address allocation system, or system built on top of protocols such as DHCPv6.

### 9.3. Role of Task scheduler

The ILA domain needs a tasks scheduler responsible for resource allocation and starting of tenant’s tasks on the ILA nodes. Defining functions of such scheduler is outside of scope of this document. At the very minimum, the scheduler would need agents running on every ILA host, participating in ILA address allocation, and communicating with the ILA control plane to publish and remove the mappings. Since it’s the scheduler that is responsible for task movements, it makes sense for the scheduler to update the mappings in the domain.

The scheduler needs some kind of API to interact with the BGP process on the box. Defining the exact API is outside of scope of this document, but as an option the scheduler may use a BGP session to inject prefixes into the BGP process running on the box.

## 10. ILA domain federation

In default operation mode, the ILA domains act as if the other domain is unaware of mappings that exist in another. It is possible to let the two domains exchange the mapping information and honor the ILA redirect messages from another domain by "merging" full or partial mapping tables of the two domains. For example, one can envision multiple compute clusters, each being its own ILA domain. In standard ILA model, those clusters would need to communicate via the ILA routers only, increasing stress on the data-plane. To allow traffic flowing directly between the hosts in each cluster and bypassing the ILA routers, the ILA domains may exchange the mapping information, and program the ILA mappings in ILA hosts to facilitate direct paths.

Since each domain may re-use the 64-bit identifier space on its own, the use of SIR prefix is required to make the identifiers globally unique. This requirement is easily fulfilled since the SIR prefix is required to be globally routable in the Internet.

To enable ILA domain federation, the BGP route-reflectors in each domain need need to be fully meshed and configured to use the "VPN-ILA" SAFI with "ILA AFI" (see [I-D.lapukhov-bgp-ila-afi]). This will propagate the mappings known to each route-reflector scoped with the SIR prefix of the local domain. If multiple domains are federated in this way, intermediate route-reflectors could be used, and filtering techniques such as described in [RFC5291] and [RFC4684] could be employed. The filtering may be further used to allow leaking of only select mappings, e.g. for the identifiers or tenants that carry lots of traffic.

If "push" distribution model is chosen with ILA domain federation, the ILA hosts will need to be configured to use "VPN-ILA" SAFI on their peering sessions with the BGP route reflectors. The ILA mapping entries lookup then need to be keyed both on the SIR prefix and the identifier to be resolved. Given the large volume of mappings that may exist in federated model, the "pull" model might become more preferable.

## 11. Operational Considerations

ILA introduces additional step in packet routing and thus adds more complexity to network troubleshooting process. At the same time, relative to the virtualization techniques that employ encapsulation and tunneling, ILA makes the underlying physical network fully visible to the tasks, and thus make tenant-driven troubleshooting simpler. This section discusses some operational procedures specific

to ILA and the additional fault models that are possible in presence of ILA.

### 11.1. Operational procedures for ILA routers

ILA routers may be added/removed from the network at any time. Adding a router is commonly needed to scale the capacity of the ILA router group when peak loads increases. Adding an ILA router is non-disruptive procedure. It starts by configuring the ILA router to peer with the BGP mesh to learn of all mappings in the domain. The use of BGP graceful restart (see [RFC4724]) would allow the new router to learn when all mappings have been advertised. At this time, the router may inject the SIR prefix, joining the operational group of ILA routers and start forwarding ILA traffic.

To gracefully take the ILA router out of service, it may be instructed to stop announcing the SIR prefix, or, in case of BGP, announce it with less preferable path attributes. This will allow the router to still accept and forward all in-flight packets, but will redirect the remaining packets toward the remaining ILA routers.

### 11.2. ICMPv6 Message generation by transit devices

Upong some conditions the transit, ILA-unware devices, may need to generate ICMPv6 messages, e.g. when IPv6 hop limit exceedes. The source of the packet sent by an ILA application would have SIR as the prefix, and hence the ICMPv6 message will need to transit an ILA router before getting back to the host that sent the original packet. This has some operational downside, as it adds path stretch to the control message flow, and needs to be accounted for operational reasons.

When an ICMPv6 message generated by an intermediate device arrives back to the sender of the original packet, the ILA may need to translate the payload of the ICMPv6 message, as it often contain the IPv6 header of the original packet. This is needed so that the control message could be properly correlated to transport level connection. Thus, it is expected that the ILA host stack will be able to perform this translation, and replace the ILA locator with SIR prefix in the destination address field of the encapsulated IPv6 header.

The last case is generating ICMPv6 message by transint device for packet sourced by non-ILA host (or outside of local ILA domain) and translated by an ILA router. In this case, the response will be directed back to the non-ILA host, bypassing the ILA router, and there will be no easy way to perform the translation of the location

portion in ILA destination address back to the SIR prefix. The non-ILA sender would be able to process the ICMPv6 message.

### 11.3. Multicast routing

Defining multicast routing and group membership dissemination is outside of scope of this document.

### 11.4. Potential ILA mapping table complications

Every packet egressing from an ILA host and matching the SIR prefix is subject to lookup and translation in the local ILA mapping table. If entry is not found, the packet is forwarded to the ILA router(s) by the virtue of SIR prefix injected in the datacenter network. If the ILA router does not have the mapping, either the ICMPv6 "Destination Unreachable" or "ILA mapping not found" message will be sent back, depending on whether the original sender is ILA or non-ILA host. There are few observations to make here:

- Packets egressing the ILA host and not matching the SIR prefix are routed as usual.
- ILA destinations that are not yet present in the ILA mapping table will be initially routed toward the ILA routers (e.g. the ILA routers will show up in the initial "traceroute" command output).
- In case of missing identifier mapping, it’s the ILA router that informs the sender of this event via either an "ILA Mapping not Found" or ICMPv6 "Destination Unreachable" messages.

Thus, the case of missing mapping is easily debuggable, though the "transition period" when the mapping is not yet in the ILA mapping table might confuse the operator using the "traceroute" command.

The most difficult case of ILA mapping table malfunction would be presence of incorrect mapping, i.e mappings pointing to a non-existent or incorrect locator.

- Non-existent locator. This will route the packet through the network, and eventually result either in packet getting discarded due to missing route or IPv6 NDP entry, or packet dropped due to routing loop and hop-limit expiration. In either case, the original sender may detect this condition either via reception of ICMPv6 "Destination Unreachable" messages, or by observing the output of the "traceroute" command. The ILA host may also be configured to make sure the identifiers fall within the known prefix range.

- Incorrect locator. In this case, the packet will be delivered to the wrong ILA host, that does not have the mapping for the identifier. Depending on whether the sending of ILA redirect messages is enabled on the host, two scenarios are possible:

  * The destination ILA host sends back an ILA redirect message with empty locator, informing the sender that mapping is invalid. The sender will invalidate the ILA mapping entry and switch over to forwarding via the ILA routers. The latter will either inform if of the new mapping, or send an ICMPv6 "Destination Unreachable" message back.

  * The destination ILA host is not configured to send the ILA redirect messages back. In this case, it simply responds with the ICMPv6 "Destination Unreachable" messages for the duration of time the sender keeps sending the packets using the incorrect mapping. The mapping needs to be flushed our updated by some external mean.

Next possible failure is dropped ILA redirect messages. However, given that the ILA redirect message sending process is memoryless, the recipient will eventually receive one of them, or at least finish the communication via an ILA router.

### 11.5. Potential ILA routers complications

The ILA routers serve as proxies for traffic entering the ILA domain, as well as temporary transit hops for traffic between the ILA hosts when they don’t have matching mappings, in case if "pull" distribution model is utilized. The following operational observations apply:

- Traffic between the ILA domain and external world will necessarily flow asymmetrically. The packets toward the ILA hosts sent from the outside will always cross the ILA routers (see Section 10 for exceptions from this case) and traffic returning from the ILA hosts to the external world will flow directly, bypassing the ILA routers. This will show up in the outputs of the "traceroute" command running from sender and destination and showing asymmetric paths. This being said, asymmetric traffic flows are very common in modern networks, and thus it should be a problem on its own.

- A failure of ILA router should be handled by re-balancing the load automatically by means of ECMP re-hashing in the network, and therefore should be mostly transparent to the ILA hosts, unless the load increases significantly after the failure. It is possible to have cascading failure and lose all ILA routers, or have them over-utilized. This event should be detected by

external monitoring system, and be acted upon by adding more ILA routers to the domain - either automatically or manually. From troubleshooting perspective, the event will manifest itself via massive packet loss toward all hosts in the ILA domain.

- o A malfunction of single ILA router (e.g. network interface card issue) would manifest itself in somewhat increased packet drop ratios for flows crossing the ILA routers, mostly traffic from external nodes. The more ILA routers the domain has, the harder to notice this ratio would be, since ECMP mostly spreads traffic evenly over all the ILA routers. This problem is more specific to ECMP behavior, and tooling exists to deal with it in datacenter networks.

- o ILA routers are in path of the ICMPv6 messages generaed by non-ILA aware routers in the network. Thus, a loss of such packet in the network could not be differentiated from the loss due to the drop by an ILA router. This may potentially complicate network troubleshooting efforts.

To sum the above up - the health of ILA router is critical to the ILA domain functions, even if "push" model is employed and the ILA routers are used mostly for external communications. The ILA routers should be monitored closely for vital parameters, such as CPU and memory utilization, traffic rates on their network interfaces, and packet loss toward the ILA routers themselves.

## 12. Deployment Scenario Primer

Building upon the concepts presented above, this section provides a simple ILA deployment scenario.

- o For locator addressing, unique-local addresses is used, with 16-bit available for sub-allocation. This allows for 1024 (2^10) Tier-3 switches with 64 (2^4) servers under each Tier-3 switch. Using the Clos topology from section Section 4.1 one can build 32 clusters with 32 Tier-3 switches each.

- o The hosts in the network would use BGP to peer with Tier-3 switches and inject their locator prefixes. It’s desirable, but not necessary to configure the route summarization on the network switches, depending on the size of the deployment.

- o Given the small to moderate scale of deployment, four IBGP route-reflectors would be deployed in the ILA domain, without the need for extra level of aggregation hierarchy. Each route-reflector will need to be configured to accept the BGP sessions from all of ILA hosts and be able to maintain thousands of peering sessions.

- The ILA hosts and routers should be configured with a single SIR prefix, and set up for "push" mapping distribution model, by disabling sending the ILA redirect messages. All ILA mappings will be propagated to all hosts and ILA routers via BGP. Each ILA host and router will need to be running a BGP process and peer with all four route-reflectors.

- The ILA routers will inject the SIR prefix using BGP into the network.

- For tasks running on ILA hosts, the globally unique ILA identifiers should be allocated independently in pseudo-random fashion by the host that first starts the task.

- As task is moved, the task scheduler will update the mapping and publish it via BGP, forcing the ILA routers and ILA hosts to update their ILA mapping tables.

- ILA domain federation is not used, making every ILA domain communicate to each other via the ILA routers only.

## 13. IANA Considerations

None

## 14. Manageability Considerations

ILA requires both one-time deployment efforts, and recurring management work. The initial involvement is reasonably high, as it required extending the existing network and host configuration. It does not require any significant changes to the existing applications, though, aside from making the applications use newly allocated IPv6 addresses. Majority of the required changes could be done without any disruption to the existing infrastructure.

ILA address management schemes could be arbitrarily complex, but in the most basic form do not require any centralized coordination. Thus, in many cases it could be a simple local subroutine that generates a pseudo-random identifier.

Recurring management efforts are mostly concentrated on monitoring the component of ILA deployment, primarily the ILA routers and the BGP route reflectors. Troubleshooting these components follows the standard process and uses regular tooling, with the caveat of having more logical components to deal with, primarily the ILA routers and the ILA mapping tables on the ILA hosts. This increases the complexity of troubleshooting process, as more state needs to be inspected and validated.

## 15. Security Considerations

ILA introduces new security considerations described below.

### 15.1. ILA host security

If unsecured ILA redirect messages are used, the ILA hosts could be exposed to cache poisoning attacks. This calls for ILA redirect message authentication, e.g. by use of digital signatures, such as [ED25519]. This will also require to use some mechanism for propagation of public keys associated with the SIR prefix (the ILA routers) and every locator in the domain, since the ILA redirect message could be sent by either.

To prevent tasks from every being able to sent packets directly bypassing the mapping layer, the ILA hosts should prohibit the task from sending packets toward the address space associated with the locators. Given that all locators will likely to belong to one large prefix, this could be accomplished by installing a single filtering rule on the ILA host.

### 15.2. BGP Security

Standard means of improving BGP security as described in [RFC7454] could be applied to harden the mapping dissemination system. Among them, the most important one is likely to be the "TCP Authentication Option" described in the referenced document. Notice that the BGP subsystem used to distribute the ILA mappings is not as vulnerable as the Internet BGP mesh, since it only work within the boundaries of a privately managed data-center.

### 15.3. ILA router security

ILA routers are primarily susceptible to various form of rate-based DDoS attacks. Primary concern would be overrruning the capabilities of ILA routers with too many packets sent from non-ILA hosts toward the SIR addresses, or "thundering herds" problem when ILA translation tables on the ILA hosts expire synchronously, or due to poisoning attack. Primary ways to address this concern would be closely monitoring server utilization and potentially rate-limiting packet flow to the ILA router on the upstream network device (ToR switch).

### 15.4. Tenant security

ILA does not natively isolate the tenant traffic from each other, nor from the underlying physical infrastructure. In fact, this is seen as one benefit that makes many troubleshooting processes easier. The access control then become responsibility of the tenant itself, by

employing traffic filtering rules. To this point, implementing filtering rules gets simpler if the tenant is allocated single prefix, as opposed to each task getting an unique identifier.

## 16. Acknowledgements

TBD

## 17. Informative References

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

[RFC4456] Bates, T., Chen, E., and R. Chandra, "BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)", RFC 4456, DOI 10.17487/RFC4456, April 2006, <http://www.rfc-editor.org/info/rfc4456>.

[RFC4684] Marques, P., Bonica, R., Fang, L., Martini, L., Raszuk, R., Patel, K., and J. Guichard, "Constrained Route Distribution for Border Gateway Protocol/MultiProtocol Label Switching (BGP/MPLS) Internet Protocol (IP) Virtual Private Networks (VPNs)", RFC 4684, DOI 10.17487/RFC4684, November 2006, <http://www.rfc-editor.org/info/rfc4684>.

[RFC5291] Chen, E. and Y. Rekhter, "Outbound Route Filtering Capability for BGP-4", RFC 5291, DOI 10.17487/RFC5291, August 2008, <http://www.rfc-editor.org/info/rfc5291>.

[RFC6740] Atkinson, RJ. and SN. Bhatti, "Identifier-Locator Network Protocol (ILNP) Architectural Description", RFC 6740, DOI 10.17487/RFC6740, November 2012, <http://www.rfc-editor.org/info/rfc6740>.

[RFC2791] Yu, J., "Scalable Routing Design Principles", RFC 2791, DOI 10.17487/RFC2791, July 2000, <http://www.rfc-editor.org/info/rfc2791>.

[RFC3633] Troan, O. and R. Droms, "IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6", RFC 3633, DOI 10.17487/RFC3633, December 2003, <http://www.rfc-editor.org/info/rfc3633>.

[RFC4724] Sangli, S., Chen, E., Fernando, R., Scudder, J., and Y. Rekhter, "Graceful Restart Mechanism for BGP", RFC 4724, DOI 10.17487/RFC4724, January 2007, <http://www.rfc-editor.org/info/rfc4724>.

[RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", RFC 4760, DOI 10.17487/RFC4760, January 2007, <http://www.rfc-editor.org/info/rfc4760>.

[RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <http://www.rfc-editor.org/info/rfc4786>.

[RFC6769] Raszuk, R., Heitz, J., Lo, A., Zhang, L., and X. Xu, "Simple Virtual Aggregation (S-VA)", RFC 6769, DOI 10.17487/RFC6769, October 2012, <http://www.rfc-editor.org/info/rfc6769>.

[RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <http://www.rfc-editor.org/info/rfc6830>.

[RFC7454] Durand, J., Pepelnjak, I., and G. Doering, "BGP Operations and Security", BCP 194, RFC 7454, DOI 10.17487/RFC7454, February 2015, <http://www.rfc-editor.org/info/rfc7454>.

[RFC7695] Pfister, P., Paterson, B., and J. Arkko, "Distributed Prefix Assignment Algorithm", RFC 7695, DOI 10.17487/RFC7695, November 2015, <http://www.rfc-editor.org/info/rfc7695>.

[RFC7938] Lapukhov, P., Premji, A., and J. Mitchell, Ed., "Use of BGP for Routing in Large-Scale Data Centers", RFC 7938, DOI 10.17487/RFC7938, August 2016, <http://www.rfc-editor.org/info/rfc7938>.

[I-D.herbert-nvo3-ila] Herbert, T., "Identifier-locator addressing for IPv6", draft-herbert-nvo3-ila-03 (work in progress), October 2016.

[I-D.lapukhov-bgp-opaque-signaling] Lapukhov, P., Aries, E., Marques, P., and E. Nkposong, "Use of BGP for Opaque Signaling", draft-lapukhov-bgp-opaque-signaling-02 (work in progress), April 2016.

[I-D.ietf-v6ops-dc-ipv6]
Lopez, D., Chen, Z., Tsou, T., Zhou, C., and A. Servin, "IPv6 Operational Guidelines for Datacenters", draft-ietf-v6ops-dc-ipv6-01 (work in progress), February 2014.

[I-D.lapukhov-bgp-ila-afi]
Lapukhov, P., "Use of BGP for dissemination of ILA mapping information", draft-lapukhov-bgp-ila-afi-01 (work in progress), March 2016.

[I-D.ietf-grow-bmp]
Scudder, J., Fernando, R., and S. Stuart, "BGP Monitoring Protocol", draft-ietf-grow-bmp-17 (work in progress), January 2016.

[I-D.ietf-nvo3-arch]
Black, D., Hudson, J., Kreeger, L., Lasserre, M., and T. Narten, "An Architecture for Data Center Network Virtualization Overlays (NVO3)", draft-ietf-nvo3-arch-08 (work in progress), September 2016.

[ED25519] "Ed25519: high-speed high-security signatures", <https://ed25519.cr.yp.to>.

[ETCD] "coreos/etcd", <https://github.com/coreos/etcd>.

[MEMCACHED]
"Memcached", <https://memcached.org/>.

[ROUTED-DESIGN]
"High Availability Campus Network Design", 2008, <http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Campus/routed-ex.html>.

[LINUX-NAMESPACES]
"Namespaces in operation, part 1: namespaces overview", 2013, <https://lwn.net/Articles/531114/>.

[IPVLAN] "IPVLAN Driver HOWTO", 2013, <https://github.com/torvalds/linux/blob/master/Documentation/networking/ipvlan.txt>.

Author’s Address

Petr Lapukhov
Facebook
1 Hacker Way
Menlo Park, CA 94025
US

Email: petr@fb.com

Routing Area Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

P. Sarkar, Ed.
Individual
S. Hegde
C. Bowers
Juniper Networks, Inc.
U. Chunduri, Ed.
Ericsson Inc.
J. Tantsura
Individual
B. Decraene
Orange
H. Gredler
Unaffiliated

July 8, 2016

# LFA selection for Multi-Homed Prefixes
# draft-psarkar-rtgwg-multihomed-prefix-lfa-04

## Abstract

This document shares experience gained from implementing algorithms to determine Loop-Free Alternates for multi-homed prefixes. In particular, this document provides explicit inequalities that can be used to evaluate neighbors as a potential alternates for multi-homed prefixes. It also provides detailed criteria for evaluating potential alternates for external prefixes advertised by OSPF ASBRs.

## Requirements Language

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

## Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2017.



Table of Contents

## 1. Introduction

The use of Loop-Free Alternates (LFA) for IP Fast Reroute is specified in [RFC5286]. Section 6.1 of [RFC5286] describes a method to determine loop-free alternates for a multi-homed prefixes (MHPs). This document describes a procedure using explicit inequalities that can be used by a computing router to evaluate a neighbor as a potential alternate for a multi-homed prefix. The results obtained are equivalent to those obtained using the method described in Section 6.1 of [RFC5286]. However, some may find this formulation useful.

Section 6.3 of [RFC5286] discusses complications associated with computing LFAs for multi-homed prefixes in OSPF. This document provides detailed criteria for evaluating potential alternates for external prefixes advertised by OSPF ASBRs, as well as explicit inequalities.

This document also provide clarifications, additional considerations to [RFC5286], to address a few coverage and operational observations. These observations are in the area of handling IS-IS attach (ATT) bit in Level-1 (L1) area, links provisioned with MAX_METRIC for traffic engineering (TE) purposes and in the area of Multi Topology (MT) IGP deployments. All these are elaborated in detail in Section 3.2 and Section 5.

### 1.1. Acronyms

AF - Address Family

ATT - IS-IS Attach Bit

ECMP - Equal Cost Multi Path

IGP - Interior Gateway Protocol

IS-IS - Intermediate System to Intermediate System

OSPF - Open Shortest Path First

MHP - Multi-homed Prefix

MT - Multi Topology

SPF - Shortest Path First PDU

## 2. LFA inequalities for MHPs

This document proposes the following set of LFA inequalities for selecting the most appropriate LFAs for multi-homed prefixes (MHPs). They can be derived from the inequalities in [RFC5286] combined with the observation that D_opt(N,P) = Min (D_opt(N,PO_i) + cost(PO_i,P)) over all PO_i

Link-Protection:
D_opt(N,PO_i)+ cost(PO_i,P) < D_opt(N,S) +
D_opt(S,PO_best) + cost(PO_best,P)

Link-Protection + Downstream-paths-only:
D_opt(N,PO_i)+ cost(PO_i,P) < D_opt(S,PO_best) + cost(PO_best,P)

Node-Protection:
D_opt(N,PO_i)+ cost(PO_i,P) < D_opt(N,E) +
D_opt(E,PO_best) + cost(PO_best,P)

Where,

| | |
|---|---|
| S | - The computing router |
| N | - The alternate router being evaluated |
| E | - The primary next-hop on shortest path from S to prefix P. |
| PO_i | - The specific prefix-originating router being evaluated. |
| PO_best | - The prefix-originating router on the shortest path from the computing router S to prefix P. |
| Cost (X,P) | - Cost of reaching the prefix P from prefix originating node X. |
| D_opt(X,Y) | - Distance on the shortest path from node X to node Y. |

Figure 1: LFA inequalities for MHPs

## 3. LFA selection for the multi-homed prefixes

To compute a valid LFA for a given multi-homed prefix P, through an alternate neighbor N a computing router S MUST follow one of the appropriate procedures below.

```
Link-Protection :
=================
1. If alternate neighbor N is also prefix-originator of P,
   1.a. Select N as a LFA for prefix P (irrespective of
        the metric advertised by N for the prefix P).
2. Else, evaluate the link-protecting LFA inequality for P with
   the N as the alternate neighbor.
   2.a. If LFA inequality condition is met,
        select N as a LFA for prefix P.
   2.b. Else, N is not a LFA for prefix P.

Link-Protection + Downstream-paths-only :
=========================================
1. Evaluate the link-protecting + downstream-only LFA inequality
   for P with the N as the alternate neighbor.
   1.a. If LFA inequality condition is met,
        select N as a LFA for prefix P.
   1.b. Else, N is not a LFA for prefix P.

Node-Protection :
=================
1. If alternate neighbor N is also prefix-originator of P,
   1.a. Select N as a LFA for prefix P (irrespective of
        the metric advertised by N for the prefix P).
2. Else, evaluate the apporpriate node-protecting LFA inequality
   for P with the N as the alternate neighbor.
   2.a. If LFA inequality condition is met,
        select N as a LFA for prefix P.
   2.b. Else, N is not a LFA for prefix P.
```

Figure 2: Rules for selecting LFA for MHPs

In case an alternate neighbor N is also one of the prefix-originators of prefix P, N MAY be selected as a valid LFA for P.

However if N is not a prefix-originator of P, the computing router SHOULD evaluate one of the corresponding LFA inequalities, as mentioned in Figure 1, once for each remote node that originated the prefix. In case the inequality is satisfied by the neighbor N router S MUST choose neighbor N, as one of the valid LFAs for the prefix P.

When computing a downstream-only LFA, in addition to being a prefix-originator of P, router N MUST also satisfy the downstream-only LFA inequality specified in Figure 1.

For more specific rules please refer to the later sections of this document.

### 3.1. Improved coverage with simplified approach to MHPs

LFA base specification [RFC5286] Section 6.1 recommends that a router compute the alternate next-hop for an IGP multi-homed prefix by considering alternate paths via all routers that have announced that prefix and the same has been elaborated with appropriate inequalities in the above section. However, [RFC5286] Section 6.1 also allows for the router to simplify the multi-homed prefix calculation by assuming that the MHP is solely attached to the router that was its pre-failure optimal point of attachment, at the expense of potentially lower coverage. If an implementation chooses to simplify the multi-homed prefix calculation by assuming that the MHP is solely attached to the router that was its pre-failure optimal point of attachment, the procedure described in this memo can potentially improve coverage for equal cost multi path (ECMP) MHPs without incurring extra computational cost.

While the approach as specified in [RFC5286] Section 6.1 last paragraph, is to simplify the MHP as solely attached to the router that was its pre-failure optimal point of attachment; though it is a scalable approach and simplifies computation, [RFC5286] notes this may result in little less coverage.

This memo improves the above approach to provide loop-free alternatives without any additional cost for equal cost multi path MHPs as described through the below example network. The approach specified here MAY also be applicable for handling default routes as explained in Section 3.2.

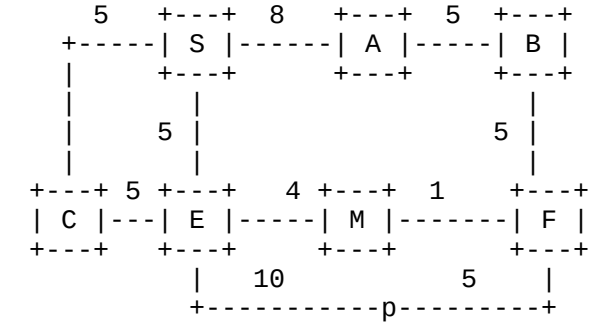



Figure 3: MHP with same ECMP Next-hop

In the above network a prefix p, is advertised from both Node E and Node F. With simplified approach taken as specified in [RFC5286] Section 6.1, prefix p will get only link protection LFA through the neighbor C while a node protection path is available through neighbor

A. In this scenario, E and F both are pre-failure optimal points of attachment and share the same primary next-hop. Hence, an implementation MAY compare the kind of protection A provides to F (link-and-node protection) with the kind of protection C provides to E (link protection) and inherit the better alternative to prefix p and here it is A.

However, in the below network prefix p has an ECMP through both node E and node F with cost 20. Though it has 2 pre-failure optimal points of attachment, the primary next-hop to each pre-failure optimal point of attachment is different. In this case, prefix p shall inherit corresponding LFA to each primary next-hop calculated for the router advertising the same respectively (node E's and node F's LFA).

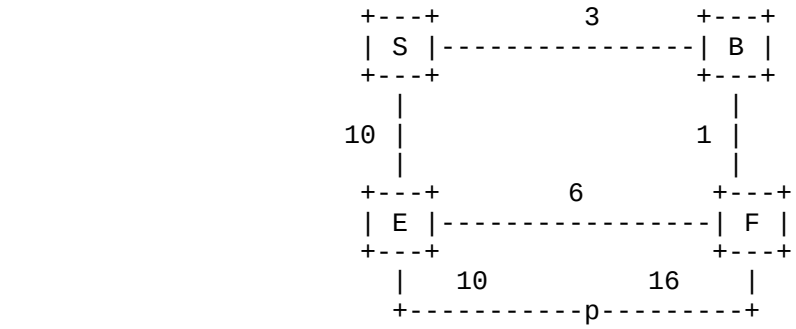



Figure 4: MHP with different ECMP Next-hops

In summary, if there are multiple pre-failure points of attachment for a MHP and primary next-hop of a MHP is same as that of the primary next-hop of the router that was pre-failure optimal point of attachment, an implementation MAY provide the better protection to MHP without incurring any additional computation cost.

## 3.2. IS-IS ATT Bit considerations

Per [RFC1195] a default route needs to be added in Level1 (L1) router to the closest reachable Level1/Level2 (L1/L2) router in the network advertising ATT (attach) bit in its LSP-0 fragment. All L1 routers in the area would do this during the decision process with the next-hop of the default route set to the adjacent router through which the closest L1/L2 router is reachable. The base LFA specification [RFC5286] does not specify any procedure for computing LFA for a default route in IS-IS L1 area. Potentially one MAY consider a default route is being advertised from the border L1/L2 router where ATT bit is set and can do LFA computation for the default route.

But, when multiple ECMP L1/L2 routers are reachable in an L1 area corresponding best LFAs SHOULD be given for each primary next-hop associated with default route. Considerations as specified in Section 3 and Section 3.1 are applicable for default routes, if the default route is considered as ECMP MHP.

## 4. LFA selection for the multi-homed external prefixes

Redistribution of external routes into IGP is required in case of two different networks getting merged into one or during protocol migrations. External routes could be distributed into an IGP domain via multiple nodes to avoid a single point of failure.

During LFA calculation, alternate LFA next-hops to reach the best ASBR could be used as LFA for the routes redistributed via that ASBR. When there is no LFA available to the best ASBR, it may be desirable to consider the other ASBRs (referred to as alternate ASBR hereafter) redistributing the external routes for LFA selection as defined in [RFC5286] and leverage the advantage of having multiple re-distributing nodes in the network.

### 4.1. IS-IS

LFA evaluation for multi-homed external prefixes in IS-IS is similar to the multi-homed internal prefixes. Inequalities described in sec 2 would also apply to multi-homed external prefixes as well.

### 4.2. OSPF

Loop free Alternates [RFC 5286] describes mechanisms to apply inequalities to find the loop free alternate neighbor. For the selection of alternate ASBR for LFA consideration, additional rules have to be applied in selecting the alternate ASBR due to the external route calculation rules imposed by [RFC 2328].

This document also defines the inequalities defined in RFC [5286] specifically for the alternate loop-free ASBR evaluation.

#### 4.2.1. Rules to select alternate ASBR

The process to select an alternate ASBR is best explained using the rules below. The below process is applied when primary ASBR for the concerned prefix is chosen and there is an alternate ASBR originating same prefix.

1. If RFC1583Compatibility is disabled

   1a. if primary ASBR and alternate ASBR are intra area non-backbone path go to step 2.
   1b. If primary ASBR and alternate ASBR belong to intra-area backbone and/or inter-area path go to step 2.
   1c. for other paths, skip the alternate ASBR and consider next ASBR.

2. If cost type (type1/type2) advertised by alternate ASBR same as primary
   2a. If not same skip alternate ASBR and consider next ASBR.

3. If cost type is type1
   3a. If cost is same, program ECMP
   3b. else go to step 5.

4 If cost type is type 2
   4a. If cost is different, skip alternate ASBR and consider next ASBR
   4b. If type2 cost is same, compare type 1 cost.
   4c. If type1 cost is also same program ECMP.
   4d. If type 1 cost is different go to step 5.

5. If route type (type 5/type 7)
   5a. If route type is same, check route p-bit, forwarding address field for routes from both ASBRs match. If not skip alternate ASBR and consider next ASBR.
   5b. If route type is not same, skip ASBR and consider next ASBR.

6. Apply inequality on the alternate ASBR.

Figure 5: Rules for selecting alternate ASBR in OSPF

#### 4.2.2. Multiple ASBRs belonging different area

When "RFC1583compatibility" is set to disabled, OSPF[RFC2328] defines certain rules of preference to choose the ASBRs. While selecting alternate ASBR for loop evaluation for LFA, these rules should be applied and ensured that the alternate neighbor does not loop the traffic back.

When there are multiple ASBRs belonging to different area advertising the same prefix, pruning rules as defined in RFC 2328 section 16.4.1

are applied. The alternate ASBRs pruned using above rules are not considered for LFA evaluation.

#### 4.2.3. Type 1 and Type 2 costs

If there are multiple ASBRs not pruned via rules defined in 3.2.2, the cost type advertised by the ASBRs is compared. ASBRs advertising Type1 costs are preferred and the type2 costs are pruned. If two ASBRs advertise same type2 cost, the alternate ASBRs are considered along with their type1 cost for evaluation. If the two ASBRs with same type2 as well as type1 cost, ECMP FRR is programmed. If there are two ASBRs with different type2 cost, the higher cost ASBR is pruned. The inequalities for evaluating alternate ASBR for type 1 and type 2 costs are same, as the alternate ASBRs with different type2 costs are pruned and the evaluation is based on equal type 2 cost ASBRS.

#### 4.2.4. RFC1583compatibility is set to enabled

When RFC1583Compatibility is set to enabled, multiple ASBRs belonging to different area advertising same prefix are chosen based on cost and hence are valid alternate ASBRs for the LFA evaluation.

#### 4.2.5. Type 7 routes

Type 5 routes always get preference over Type 7 and the alternate ASBRs chosen for LFA calculation should belong to same type.Among Type 7 routes, routes with p-bit and forwarding address set have higher preference than routes without these attributes. Alternate ASBRs selected for LFA comparison should have same p-bit and forwarding address attributes.

#### 4.2.6. Inequalities to be applied for alternate ASBR selection

The alternate ASBRs selected using above mechanism described in 3.2.1, are evaluated for Loop free criteria using below inequalities.

##### 4.2.6.1. Forwarding address set to non zero value

```
  Link-Protection:
  F_opt(N,PO_i)+ cost(PO_i,P) < D_opt(N,S) +
                                F_opt(S,PO_best) + cost(PO_best,P)

  Link-Protection + Downstream-paths-only:
  F_opt(N,PO_i)+ cost(PO_i,P) < F_opt(S,PO_best) + cost(PO_best,P)

  Node-Protection:
  F_opt(N,PO_i)+ cost(PO_i,P) < D_opt(N,E) +
                                F_opt(E,PO_best) + cost(PO_best,P)

  Where,
  S             - The computing router
  N             - The alternate router being evaluated
  E             - The primary next-hop on shortest path from S to
                          prefix P.
  PO_i          - The specific prefix-originating router being
                          evaluated.
  PO_best       - The prefix-originating router on the shortest path
                          from the computing router S to prefix P.
  cost(X,Y)     - External cost for Y as advertised by X
  F_opt(X,Y)    - Distance on the shortest path from node X to Forwarding
                  address specified by ASBR Y.
  D_opt(X,Y)    - Distance on the shortest path from node X to node Y.
```

Figure 6: LFA inequality definition when forwarding address in non-zero

#### 4.2.6.2. ASBRs advertising type1 and type2 cost

```
Link-Protection:
D_opt(N,PO_i)+ cost(PO_i,P) < D_opt(N,S) +
                              D_opt(S,PO_best) + cost(PO_best,P)

Link-Protection + Downstream-paths-only:
D_opt(N,PO_i)+ cost(PO_i,P) < D_opt(S,PO_best) + cost(PO_best,P)

Node-Protection:
D_opt(N,PO_i)+ cost(PO_i,P) < D_opt(N,E) +
                              D_opt(E,PO_best) + cost(PO_best,P)

Where,
S            - The computing router
N            - The alternate router being evaluated
E            - The primary next-hop on shortest path from S to
                        prefix P.
PO_i         - The specific prefix-originating router being
                        evaluated.
PO_best      - The prefix-originating router on the shortest path
                        from the computing router S to prefix P.
cost(X,Y)    - External cost for Y as advertised by X.
D_opt(X,Y)   - Distance on the shortest path from node X to node Y.
```

Figure 7: LFA inequality definition for type1 and type 2 cost

## 5. LFA Extended Procedures

This section explains the additional considerations in various aspects as listed below to the base LFA specification [RFC5286].

### 5.1. Links with IGP MAX_METRIC

Section 3.5 and 3.6 of [RFC5286] describes procedures for excluding nodes and links from use in alternate paths based on the maximum link metric (as defined in for IS-IS in [RFC5305] or as defined in [RFC3137] for OSPF). If these procedures are strictly followed, there are situations, as described below, where the only potential alternate available which satisfies the basic loop-free condition will not be considered as alternative.

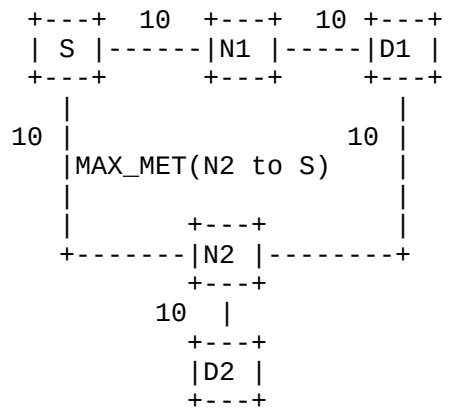



Figure 8: Link with IGP MAX_METRIC

In the simple example network, all the link costs have a cost of 10 in both directions, except for the link between S and N2. The S-N2 link has a cost of 10 in the direction from S to N2, and a cost of MAX_METRIC in the direction from N2 to S (0xffffff /2^24 - 1 for IS-IS and 0xffff for OSPF) for a specific end to end Traffic Engineering (TE) requirement of the operator. At node S, D1 is reachable through N1 with cost 20, and D2 is reachable through N2 with cost 20. Even though neighbor N2 satisfies basic loop-free condition (inequality 1 of [RFC5286]) for D1 this could be excluded as potential alternative because of the current exclusions as specified in section 3.5 and 3.6 procedure of [RFC5286]. But, as the primary traffic destined to D2 continue to use the link and hence irrespective of the reverse metric in this case, the same link MAY be used as a potential LFA for D1.

Alternatively, reverse metric of the link MAY be configured with MAX_METRIC-1, so that the link can be used as an alternative while meeting the TE requirements.

## 5.2. Multi Topology Considerations

Section 6.2 and 6.3.2 of [RFC5286] state that multi-topology OSPF and ISIS are out of scope for that specification. This memo clarifies and describes the applicability.

In Multi Topology (MT) IGP deployments, for each MT ID, a separate shortest path tree (SPT) is built with topology specific adjacencies, the LFA principles laid out in [RFC5286] are actually applicable for MT IS-IS [RFC5120] LFA SPF. The primary difference in this case is, identifying the eligible-set of neighbors for each LFA computation which is done per MT ID. The eligible-set for each MT ID is

determined by the presence of IGP adjacency from Source to the neighboring node on that MT-ID apart from the administrative restrictions and other checks laid out in [RFC5286]. The same is also applicable for OSPF [RFC4915] [MT-OSPF] or different AFs in multi instance OSPFv3 [RFC5838].

However for MT IS-IS, if a default topology is used with MT-ID 0 [RFC5286] and both IPv4 [RFC5305] and IPv6 routes/AFs [RFC5308] are present, then the condition of network congruency is applicable for LFA computation as well. Network congruency here refers to, having same address families provisioned on all the links and all the nodes of the network with MT-ID 0. Here with single decision process both IPv4 and IPv6 next-hops are computed for all the prefixes in the network and similarly with one LFA computation from all eligible neighbors per [RFC5286], all potential alternatives can be computed.

## 6. Acknowledgements

Thanks to Alia Atlas and Salih K A for their useful feedback and inputs.

## 7. IANA Considerations

N/A. - No protocol changes are proposed in this document.

## 8. Security Considerations

This document does not introduce any change in any of the protocol specifications and also this does not introduce any new security issues other than as noted in the LFA base specification [RFC5286].

## 9. References

### 9.1. Normative References

[RFC1195] Callon, R., "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, DOI 10.17487/RFC1195, December 1990, <http://www.rfc-editor.org/info/rfc1195>.

[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>.

## 9.2. Informative References

[I-D.ietf-rtgwg-lfa-manageability]
Litkowski, S., Decraene, B., Filsfils, C., Raza, K., and M. Horneffer, "Operational management of Loop Free Alternates", draft-ietf-rtgwg-lfa-manageability-11 (work in progress), June 2015.

[RFC3137] Retana, A., Nguyen, L., White, R., Zinin, A., and D. McPherson, "OSPF Stub Router Advertisement", RFC 3137, DOI 10.17487/RFC3137, June 2001, <http://www.rfc-editor.org/info/rfc3137>.

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

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

[RFC5286] Atlas, A., Ed. and A. Zinin, Ed., "Basic Specification for IP Fast Reroute: Loop-Free Alternates", RFC 5286, DOI 10.17487/RFC5286, September 2008, <http://www.rfc-editor.org/info/rfc5286>.

[RFC5305] Li, T. and H. Smit, "IS-IS Extensions for Traffic Engineering", RFC 5305, DOI 10.17487/RFC5305, October 2008, <http://www.rfc-editor.org/info/rfc5305>.

[RFC5308] Hopps, C., "Routing IPv6 with IS-IS", RFC 5308, DOI 10.17487/RFC5308, October 2008, <http://www.rfc-editor.org/info/rfc5308>.

[RFC5838] Lindem, A., Ed., Mirtorabi, S., Roy, A., Barnes, M., and R. Aggarwal, "Support of Address Families in OSPFv3", RFC 5838, DOI 10.17487/RFC5838, April 2010, <http://www.rfc-editor.org/info/rfc5838>.

## Authors' Addresses

Pushpasis Sarkar (editor)
Individual

Email: pushpasis.ietf@gmail.com

Shraddha Hegde
Juniper Networks, Inc.
Electra, Exora Business Park
Bangalore, KA 560103
India

Email: shraddha@juniper.net

Chris Bowers
Juniper Networks, Inc.
1194 N. Mathilda Ave.
Sunnyvale, CA 94089
US

Email: cbowers@juniper.net

Uma Chunduri (editor)
Ericsson Inc.
300 Holger Way,
San Jose, California 95134
USA

Phone: 408 750-5678
Email: uma.chunduri@ericsson.com

Jeff Tantsura
Individual

Email: jefftant.ietf@gmail.com

Bruno Decraene
Orange

Email: bruno.decraene@orange.com

Hannes Gredler
Unaffiliated

Email: hannes@gredler.at

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 27, 2017

A. Lindem, Ed.
Cisco Systems
L. Berger, Ed.
LabN Consulting, L.L.C.
D. Bogdanovic

C. Hopps
Deutsche Telekom
July 26, 2016

# Network Device YANG Organizational Models
## draft-rtgyangdt-rtgwg-device-model-05

Abstract

This document presents an approach for organizing YANG models in a comprehensive structure that may be used to configure and operate network devices. The structure is itself represented as a YANG model, with all of the related component models logically organized in a way that is operationally intuitive, but this model is not expected to be implemented. The identified component modules are expected to be defined and implemented on common network devices.

This document is derived from work submitted to the IETF by members of the informal OpenConfig working group of network operators and is a product of the Routing Area YANG Architecture design team.

Table of Contents



## 1. Introduction

"Operational Structure and Organization of YANG Models" [I-D.openconfig-netmod-model-structure], highlights the value of organizing individual, self-standing YANG [RFC6020] models into a more comprehensive structure. This document builds on that work and presents a derivative structure for use in representing the networking infrastructure aspects of physical and virtual devices. [I-D.openconfig-netmod-model-structure] and earlier versions of this document presented a single device-centric model root, this document no longer contains this element. Such an element would have translated to a single device management model that would be the root

of all other models and was judged to be overly restrictive in terms of definition, implementation, and operation.

The document presents a notional network device YANG organizational structure that provides a conceptual framework for the models that may be used to configure and operate network devices. The structure is itself presented as a YANG module, with all of the related component modules logically organized in a way that is operationally intuitive. This network device model is not expected to be implemented, but rather provide as context for the identified representative component modules with are expected to be defined, and supported on typical network devices.

This document refers to two new modules that are expected to be implemented. These models are defined to support the configuration and operation of network-devices that allow for the partitioning of resources from both, or either, management and networking perspectives. Two forms of resource partitioning are referenced:

The first form provides a logical partitioning of a network device where each partition is separately managed as essentially an independent network element which is 'hosted' by the base network device. These hosted network elements are referred to as logical network elements, or LNEs, and are supported by the logical-network-element module defined in [LNE-MODEL]. The module is used to identify LNEs and associate resources from the network-device with each LNE. LNEs themselves are represented in YANG as independent network devices; each accessed independently. Optionally, and when supported by the implementation, they may also be accessed from the host system. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

The second form provides support what is commonly referred to as Virtual Routing and Forwarding (VRF) instances as well as Virtual Switch Instances (VSI), see [RFC4026]. In this form of resource partitioning multiple control plane and forwarding/bridging instances are provided by and managed via a single (physical or logical) network device. This form of resource partitioning is referred to as Network Instances and are supported by the network-instance module defined in [NI-MODEL]. Configuration and operation of each network-instance is always via the network device and the network-instance module.

This document was motivated by, and derived from, [I-D.openconfig-netmod-model-structure]. The requirements from that document have been combined with the requirements from "Consistent Modeling of Operational State Data in YANG",

[I-D.openconfig-netmod-opstate], into "NETMOD Operational State Requirements", [I-D.ietf-netmod-opstate-reqs]. This document is aimed at the requirement related to a common model-structure, currently Requirement 7, and also aims to provide a modeling base for Operational State representation.

The approach taken in this (and the original) document is to organize the models describing various aspects of network infrastructure, focusing on devices, their subsystems, and relevant protocols operating at the link and network layers. The proposal does not consider a common model for higher level network services. We focus on the set of models that are commonly used by network operators, and suggest a corresponding organization.

A significant portion of the text and model contained in this document was taken from the -00 of [I-D.openconfig-netmod-model-structure].

### 1.1. Status of Work and Open Issues

This version of the document and structure are a product of the Routing Area YANG Architecture design team and is very much a work in progress rather than a final proposal. This version is a major change from the prior version and this change was enabled by the work on the previously mentioned Schema Mount.

Schema Mount enables a dramatic simplification of the presented device model, particularly for "lower-end" devices which are unlikely to support multiple network instances or logical network elements. Should structural-mount/YSDL not be available, the more explicit tree structure presented in earlier versions of this document will need to be utilized.

The top open issues are:

1. This document will need to match the evolution and standardization of [I-D.openconfig-netmod-opstate] or [I-D.ietf-netmod-opstate-reqs] by the Netmod WG.

2. Interpretation of different policy containers requires clarification.

3. It may make sense to use the identityref structuring with hardware and QoS model.

4. Which document(s) define the base System management, network services, and oam protocols modules is TBD. This includes the

possibility of simply using RFC7317 in place of the presented System management module.

5. The model will be updated once the "opstate" requirements are addressed.

## 2. Module Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g, routers, firewalls and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs) each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

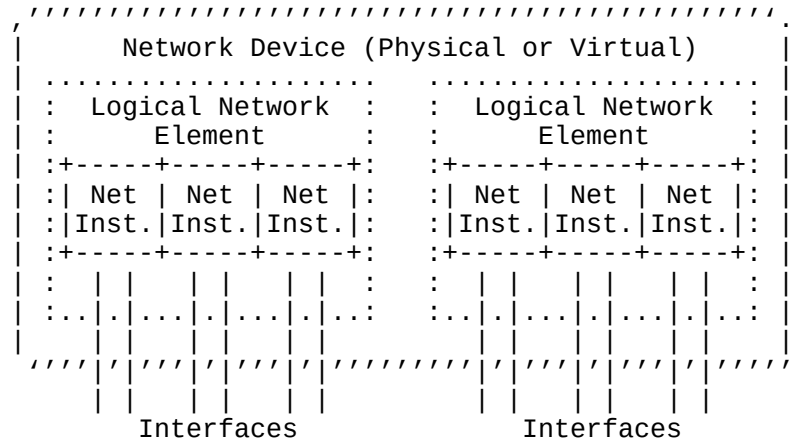



Figure 1: Module Element Relationships

A model for LNEs is described in [LNE-MODEL] and the model for network instances is covered in [NI-MODEL].

The presented notional network device module can itself be thought of as a "meta-model" as it describes the relationships between individual models. We choose to represent it also as a simple YANG module consisting of other models, which are in fact independent top

level individual models. Although it is never expected to be implemented.

The presented modules do not follow the hierarchy of any Particular implementation, and hence is vendor-neutral. Nevertheless, the structure should be familiar to network operators and also readily mapped to vendor implementations.

The overall structure is:

```
module: ietf-network-device
   +--rw modules-state             [I-D.ietf-netconf-yang-library]
   |
   +--rw interfaces                [RFC7223]
   +--rw hardware
   +--rw qos
   |
   +--rw system-management         [RFC7317 or derived]
   +--rw network-services
   +--rw oam-protocols
   |
   +--rw routing                   [I-D.ietf-netmod-routing-cfg]
   +--rw mpls
   +--rw ieee-dot1Q
   |
   +--rw acls                      [I-D.ietf-netmod-acl-model]
   +--rw key-chains                [I-D.ietf-rtgwg-yang-key-chain]
   |
   +--rw logical-network-elements [I-D.rtgyangdt-rtgwg-lne-model]
   +--rw network-instances         [I-D.rtgyangdt-rtgwg-ni-model]
```

The network device is composed of top level modules that can be used to configure and operate a network device. (This is a significant difference from earlier versions of this document where there was a strict model hierarchy.) Importantly the network device structure is the same for a physical network device or a logical network device, such as those instantiated via the logical-network-element model. Extra spacing is included to denote different types of modules included.

YANG library [I-D.ietf-netconf-yang-library] is included as it used to identify details of the top level modules supported by the (physical or logical) network device. Th ability to identify supported modules is particularly important for LNEs which may have a set of supported modules which differs from the set supported by the host network device.

The interface management model [RFC7223] is included at the top level. The hardware module is a placeholder for a future device-specific configuration and operational state data model. For example, a common structure for the hardware model might include chassis, line cards, and ports, but we leave this unspecified. The quality of service (QoS) section is also a placeholder module for device configuration and operational state data which relates to the treatment of traffic across the device. This document references augmentations to the interface module to support LNEs and NIs. Similar elements, although perhaps only for LNEs, may also need to be included as part of the definition of the future hardware and QoS modules.

System management, network services, and oam protocols represent new top level modules that are used to organize data models of similar functions. Additional information on each is provided below.

The routing and MPLS modules provide core support for the configuration and operation of a devices control plane and data plane functions. IEEE dot1Q [IEEE-8021Q] is an example of another module that provides similar functions for VLAN bridging, and other similar modules are also possible. Each of these modules is expected to be LNE and NI unaware, and to be instantiated as needed as part of the LNE and NI configuration and operation supported by the logical-network-element and network-instance modules. (Note that this is a change from [I-D.ietf-netmod-routing-cfg] which is currently defined with VRF/NI semantics.)

The access control list (ACL) and key chain modules are included as examples of other top level modules that may be supported by a network device.

The logical network element and network instance modules enable LNEs and NIs respectively and are defined below.

## 2.1. Interface Model Components

Interfaces are a crucial part of any network device’s configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [RFC7223].

The logical-network-element and network-instance modules defined in [LNE-MODEL] and [NI-MODEL] augment the existing interface management model in two ways: The first, by the logical-network-element module, adds an identifier which is used on physical interface types to identify an associated LNE. The second, by the network-instance module, adds a name which is used on interface or sub-interface types to identify an associated network instance. Similarly, this name is also added for IPv4 and IPv6 types, as defined in [RFC7277].

The interface related augmentations are as follows:

```
module: ietf-logical-network-element
augment /if:interfaces/if:interface:
   +--rw bind-lne-name?   string

module: ietf-network-instance
augment /if:interfaces/if:interface:
   +--rw bind-network-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv4:
   +--rw bind-network-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv6:
   +--rw bind-network-instance-name?   string
```

The following is an example of envisioned combined usage. The interfaces container includes a number of commonly used components as examples:

```
            +--rw if:interfaces
            |  +--rw interface* [name]
            |     +--rw name                        string
            |     +--rw lne:bind-lne-name?          string
            |     +--rw ethernet
            |     |  +--rw ni:bind-network-instance-name? string
            |     |  +--rw aggregates
            |     |  +--rw rstp
            |     |  +--rw lldp
            |     |  +--rw ptp
            |     +--rw vlans
            |     +--rw tunnels
            |     +--rw ipv4
            |     |  +--rw ni:bind-network-instance-name? string
            |     |  +--rw arp
            |     |  +--rw icmp
            |     |  +--rw vrrp
            |     |  +--rw dhcp-client
            |     +--rw ipv6
            |        +--rw ni:bind-network-instance-name? string
            |        +--rw vrrp
            |        +--rw icmpv6
            |        +--rw nd
            |        +--rw dhcpv6-client
```

The [RFC7223] defined interface model is structured to include all interfaces in a flat list, without regard to logical or virtual instances (e.g., VRFs) supported on the device. The bind-lne-name and bind-network-instance-name leaves provide the association between an interface and its associated LNE and NI (e.g., VRF or VSI).

## 2.2. System Management

[Editor’s note: need to discuss and resolve relationship between this structure and RFC7317 and determine if 7317 is close enough to simply use as is.]

System management is expected to reuse definitions contained in [RFC7317]. It is expected to be instantiated per device and LNE. Its structure is shown below:

```
    module: ietf-network-device
       +--rw system-management
       |  +--rw system-management-global
       |  +--rw system-management-protocol* [type]
       |     +--rw type    identityref
```

System-management-global is used for configuration information and state that is independent of a particular management protocol. System-management-protocol is a list of management protocol specific elements. The type-specific sub-modules are expected to be defined.

The following is an example of envisioned usage:

```
module: ietf-network-device
   +--rw system-management
      +--rw system-management-global
      |  +--rw statistics-collection
      |  ...
      +--rw system-management-protocol* [type]
      |  +--rw type=syslog
      |  +--rw type=dns
      |  +--rw type=ntp
      |  +--rw type=ssh
      |  +--rw type=tacacs
      |  +--rw type=snmp
      |  +--rw type=netconf
```

## 2.3. Network Services

A device may provide different network services to other devices, for example a device my act as a DHCP server. The model may be instantiated per device, LNE, and NI. An identityref is used to identify the type of specific service being provided and its associated configuration and state information. The defined structure is as follows:

```
module: ietf-network-device
   +--rw network-services
   |  +--rw network-service* [type]
   |     +--rw type    identityref
```

The following is an example of envisioned usage: Examples shown below include a device-based Network Time Protocol (NTP) server, a Domain Name System (DNS) server, and a Dynamic Host Configuration Protocol (DHCP) server:

```
module: ietf-network-device
   +--rw network-services
      +--rw network-service* [type]
         +--rw type=ntp-server
         +--rw type=dns-server
         +--rw type=dhcp-server
```

## 2.4. OAM Protocols

OAM protocols that may run within the context of a device are grouped within the oam-protocols model. The model may be instantiated per device, LNE, and NI. An identifyref is used to identify the information and state that may relate to a specific OAM protocol. The defined structure is as follows:

```
module: ietf-network-device
   +--rw oam-protocols
      +--rw oam-protocol* [type]
         +--rw type    identityref
```

The following is an example of envisioned usage. Examples shown below include Bi-directional Forwarding Detection (BFD), Ethernet Connectivity Fault Management (CFM), and Two-Way Active Measurement Protocol (TWAMP):

```
module: ietf-network-device
   +--rw oam-protocols
      +--rw oam-protocol* [type]
         +--rw type=bfd
         +--rw type=cfm
         +--rw type=twamp
```

## 2.5. Routing

Routing protocol and IP forwarding configuration and operation information is modeled via a routing model, such as the one defined in [I-D.ietf-netmod-routing-cfg].

The routing module is expected to include all IETF defined control plane protocols, such as BGP, OSPF, LDP and RSVP-TE. It is also expected to support configuration and operation of or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Finally, policy is expected to be represented within each control plane protocol and RIB.

The anticipated structure is as follows:

```
 module: ietf-network-device
     +--rw rt:routing                [I-D.ietf-netmod-routing-cfg]
        +--rw control-plane-protocol* [type]
        |  +--rw type      identityref
        |  +--rw policy
        +--rw rib* [name]
           +--rw name           string
           +--rw description?   string
           +--rw policy
```

## 2.6. MPLS

MPLS data plane related information is grouped together, as with the previously discussed modules, is unaware of VRFs/NIs. The model may be instantiated per device, LNE, and NI. MPLS control plane protocols are expected to be included in Section 2.5. MPLS may reuse and build on [I-D.openconfig-mpls-consolidated-model] or other emerging models and has an anticipated structure as follows:

```
  module: ietf-network-device
       +--rw mpls
          +--rw global
          +--rw lsps* [type]
             +--rw type    identityref
```

Type refers to LSP type, such as static, traffic engineered or routing congruent. The following is an example of such usage:

```
  module: ietf-network-device
       +--rw mpls
          +--rw global
          +--rw lsps* [type]
              +--rw type=static
              +--rw type=constrained-paths
              +--rw type=igp-congruent
```

# 3. Security Considerations

The network-device model structure described in this document does not define actual configuration and state data, hence it is not directly responsible for security risks.

Each of the component models that provide the corresponding configuration and state data should be considered sensitive from a security standpoint since they generally manipulate aspects of network configurations. Each component model should be carefully evaluated to determine its security risks, along with mitigations to reduce such risks.

LNE portion is TBD

NI portion is TBD

## 4. IANA Considerations

This YANG model currently uses a temporary ad-hoc namespace. If it is placed or redirected for the standards track, an appropriate namespace URI will be registered in the "IETF XML Registry" [RFC3688]. The YANG structure modules will be registered in the "YANG Module Names" registry [RFC6020].

## 5. Network Device Model Structure

```
<CODE BEGINS> file "ietf-network-device@2016-05-01.yang"
module ietf-network-device {

  yang-version "1";

  // namespace
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-device";

  prefix "nd";

  // import some basic types

  // meta
  organization "IETF RTG YANG Design Team Collaboration
                with OpenConfig";

  contact
      "Routing Area YANG Architecture Design Team -
       <rtg-dt-yang-arch@ietf.org>";

  description
    "This module describes a model structure for YANG
     configuration and operational state data models. Its intent is
     to describe how individual device protocol and feature models
     fit together and interact.";

  revision "2016-05-01" {
    description
      "IETF Routing YANG Design Team Meta-Model";
    reference "TBD";
  }

  // extension statements
```

```
     // identity statements

     identity oam-protocol-type {
         description
             "Base identity for derivation of OAM protocols";
     }

     identity network-service-type {
         description
             "Base identity for derivation of network services";
     }

      identity system-management-protocol-type {
         description
             "Base identity for derivation of system management
              protocols";
      }

      identity oam-service-type {
         description
             "Base identity for derivation of Operations,
              Administration, and Maintenance (OAM) services.";
      }

      identity control-plane-protocol-type {
         description
             "Base identity for derivation of control-plane protocols";
      }

      identity mpls-lsp-type {
         description
             "Base identity for derivation of MPLS LSP typs";
      }

     // typedef statements

     // grouping statements

     grouping ribs {
       description
         "Routing Information Bases (RIBs) supported by a
          network-instance";
       container ribs {
           description
               "RIBs supported by a network-instance";
           list rib {
               key "name";
               min-elements "1";
```

```
            description
                "Each entry represents a RIB identified by the
               'name' key. All routes in a RIB must belong to the
                same address family.

                For each routing instance, an implementation should
                provide one system-controlled default RIB for each
                supported address family.";
            leaf name {
                type string;
                description
                    "The name of the RIB.";
            }
            reference "draft-ietf-netmod-routing-cfg";
            leaf description {
                type string;
                description
                    "Description of the RIB";
            }
            // Note that there is no list of interfaces within
            container policy {
                description "Policy specific to RIB";
            }
        }
      }
    }

    // top level device definition statements
    container ietf-yang-library {
      description
        "YANG Module Library as defined in
         draft-ietf-netconf-yang-library";
    }

    container interfaces {
      description
       "Interface list as defined by RFC7223/RFC7224";
    }

    container hardware {
      description
        "Hardware / vendor-specific data relevant to the platform.
        This container is an anchor point for platform-specific
        configuration and operational state data.  It may be further
        organized into chassis, line cards, ports, etc.  It is
        expected that vendor or platform-specific augmentations
        would be used to populate this part of the device model";
    }
```

```
container qos {
  description "QoS features, for example policing, shaping, etc.";
}

container system-management {
    description
      "System management for physical or virtual device.";
    container system-management-global {
        description "System management - with reuse of RFC 7317";
    }
    list system-management-protocol {
        key "type";
        leaf type {
            type identityref {
                base system-management-protocol-type;
            }
            mandatory true;
            description
                "Syslog, ssh, TACAC+, SNMP, NETCONF, etc.";
        }
        description "List of system management protocol
                     configured for a logical network
                     element.";
    }
}

container network-services {
    description
        "Container for list of configured network
         services.";
    list network-service {
        key "type";
        description
            "List of network services configured for a
             network instance.";
        leaf type {
            type identityref {
                base network-service-type;
            }
            mandatory true;
            description
                "The network service type supported within
                 a network instance, e.g., NTP server, DNS
                 server, DHCP server, etc.";
        }
    }
}
```

```
     container oam-protocols {
         description
             "Container for configured OAM protocols.";
         list oam-protocol {
             key "type";
             leaf type {
                 type identityref {
                     base oam-protocol-type;
                 }
                 mandatory true;
                 description
                     "The Operations, Administration, and
                      Maintenance (OAM) protocol type, e.g., BFD,
                      TWAMP, CFM, etc.";
             }
             description
                 "List of configured OAM protocols.";
         }
     }

     container routing {
       description
         "The YANG Data Model for Routing Management revised to be
          Network Instance / VRF independent. ";
       // Note that there is no routing or network instance
       list control-plane-protocol {
           key "type";
           description
               "List of control plane protocols configured for
                a network instance.";
           leaf type {
               type identityref {
                   base control-plane-protocol-type;
               }
               description
                   "The control plane protocol type, e.g., BGP,
                    OSPF IS-IS, etc";
           }
           container policy {
               description
                   "Protocol specific policy,
                   reusing [RTG-POLICY]";
           }
       }
       list rib {
           key "name";
           min-elements "1";
           description
```

```
                "Each entry represents a RIB identified by the
               'name' key. All routes in a RIB must belong to the
                same address family.

                For each routing instance, an implementation should
                provide one system-controlled default RIB for each
                supported address family.";
            leaf name {
                type string;
                description
                    "The name of the RIB.";
            }
            reference "draft-ietf-netmod-routing-cfg";
            leaf description {
                type string;
                description
                    "Description of the RIB";
            }
            // Note that there is no list of interfaces within
            container policy {
                description "Policy specific to RIB";
            }
          }
        }

        container mpls {
            description "MPLS and TE configuration";
            container global {
                description "Global MPLS configuration";
            }
            list lsps {
                key "type";
                description
                    "List of LSP types.";
                leaf type {
                    type identityref {
                        base mpls-lsp-type;
                    }
                    mandatory true;
                    description
                        "MPLS and Traffic Engineering protocol LSP types,
                         static, LDP/SR (igp-congruent),
                         RSVP TE (constrained-paths) , etc.";
                }
            }
        }

        container ieee-dot1Q {
```

```
      description
        "The YANG Data Model for VLAN bridges as defined by the IEEE";
    }

    container ietf-acl {
      description "Packet Access Control Lists (ACLs) as specified
                     in draft-ietf-netmod-acl-model";
    }

    container ietf-key-chain {
      description "Key chains as specified in
                   draft-ietf-rtgwg-yang-key-chain;";
    }

    container logical-network-element {
      description
        "This module is used to support multiple logical network
         elements on a single physical or virtual system.";
    }

    container network-instance {
      description
        "This module is used to support multiple network instances
         within a single physical or virtual device.  Network
         instances are commonly know as VRFs (virtual routing
         and forwarding) and VSIs (virtual switching instances).";
    }
    // rpc statements

    // notification statements

  }
  <CODE ENDS>
```

# 6. References

## 6.1. Normative References

[LNE-MODEL]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "Logical Network Element Model", draft-rtgyangdt-rtgwg-lne-model-00.txt (work in progress), May 2016.

[NI-MODEL]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "Network Instance Model", draft-rtgyangdt-rtgwg-ni-model-00.txt (work in progress), May 2016.

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

[RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <http://www.rfc-editor.org/info/rfc4026>.

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

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

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <http://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

## 6.2. Informative References

[I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library-05 (work in progress), April 2016.

[I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-03 (work in progress), January 2016.

[I-D.ietf-netmod-routing-cfg]
Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-20 (work in progress), October 2015.

[I-D.openconfig-mpls-consolidated-model]
George, J., Fang, L., eric.osborne@level3.com, e., and R. Shakir, "MPLS / TE Model for Service Provider Networks", draft-openconfig-mpls-consolidated-model-02 (work in progress), October 2015.

[I-D.openconfig-netmod-model-structure]
Shaikh, A., Shakir, R., D'Souza, K., and L. Fang, "Operational Structure and Organization of YANG Models", draft-openconfig-netmod-model-structure-00 (work in progress), March 2015.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[IEEE-8021Q]
Holness, M., "IEEE 802.1Q YANG Module Specifications", IEEE-Draft http://www.ieee802.org/1/files/public/docs2015/new-mholness-yang-8021Q-0515-v04.pdf, May 2015.

## Appendix A. Acknowledgments

This document is derived from draft-openconfig-netmod-model-structure-00. We thank the Authors of that document and acknowledge their indirect contribution to this work. The authors include: Anees Shaikh, Rob Shakir, Kevin D'Souza, Luyuan Fang, Qin Wu, Stephane Litkowski and Gang Yan.

This work was discussed in and produced by the Routing Area Yang Architecture design team. Members at the time of writing included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Gang Yan.

The identityref approach was proposed by Mahesh Jethanandani.

The RFC text was produced using Marshall Rose's xml2rfc tool.

Authors' Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Lou Berger (editor)
LabN Consulting, L.L.C.

Email: lberger@labn.net

Dean Bogdanovic

Email: ivandean@gmail.com

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org