                       I2RS Ephemeral Datastore
                   draft-hares-i2rs-ephemeral-ds-00.txt

Abstract

   This document the Yang module for the I2RS ephemeral datastore.

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 May 16, 2018.

Table of Contents

1. Introduction

   The I2RS architecture [RFC7921] defines the I2RS interface "a
   programmatic interface for state transfer in and out of the Internet
   routing system". The I2RS interface consists of the I2RS ephemeral
   dynamic datastore populated with modules which operate within that
   ephemeral datastore plus a protocol to access this datastore. This
   document provides the yang for the I2RS ephemeral dynamic datastore
   as a basic for developers who wish to populate it with specific
   modules.

   The protocol which access the ephemeral datastore is an IETF
   management protocol (NETCONF [RFC6241], RESTCONF [RFC8040]) which
   have been extended in the revised data stores module
   ([I-D.ietf-netconf-nmda-restconf], [I-D.ietf-netconf-nmda-netconf]).
   These basic protocols meet the I2RS requirements for ephemeral state
   [RFC8242] and protocol security [RFC8241].

2. Definitions

2.1. 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].

2.2. I2RS Definitions

   The I2RS architecture [RFC7921] defines the following:

   ephemeral data: is data which does not persist across a reboot
      (software or hardware) or a power on/off condition. Ephemeral
      data can be configured data or data recorded from operations of
      the router. Ephemeral configuration data also has the property
      that a system cannot roll back to a previous ephemeral
      configuration state. (See [RFC7921] for an architectural
      overview, [RFC8242] for requirements, and
      [I-D.ietf-netmod-revised-datastores] for discussion of how the
      ephemeral datastore as a dynamic datastore interacts with intended
      configuration datstore, the dynamic configuration protocols, and
      control planes datastore to create the applied datastore and
      operational state datastore.

3. Operational Options

   The I2RS ephemeral datastore requires the revised datastores
   ([I-D.ietf-netmod-revised-datastores]).

   It is suggested that any implementation provide the following
   operator options as "knobs" the operator can set:

   o Knobs to determine whether local policy or I2RS has precedence.

   o Knob for allowing only secure transport (e.g. TLS) or allowing
      both secure and insecure transport. Insecure transport can only
      support for items denoted as "data-not-sensitive" in the module.
      The recommend default setting for this knob is not allow insecure
      transport.

4. Publishing non-Secure Data

   Non-Secure data may be published from an I2RS datastream as a stream
   of notifications or a set of data read. For example, if the routing
   system attaches to a web site which is up via multiple links, the
   I2RS may want to publish the availability or non-availability of such
   a web site via a notification stream. In this case, the notification
   stream in RESTCONF might run over HTTP over TCP instead of HTTP over
   TLS.

   Any data module which uses this feature should undergo additional
   security review to determine that this non-secure stream does not
   provide an additional attack surface. Any yang data module being
   standardized in the IETF which utilizes non-secure data should be
   reviewed by IETF experts in routing, operations, and security to
   determine if the non-secure data provides an acceptable mitigation of
   security risks.

5. Yang for Ephemeral Datastore

     1. Name   : ephemeral
     2. YANG modules   : all (default)
     3. YANG statements : config false + ephemeral true
     4. How applied   : automatic
     5. Protocols   : NC/RC (default)
     6. YANG Module   : (see below)
         7. Ephemeral-capable modules: (see IANA registry)
         8. illegal features: (features illegal for I2RS datastore
         9. Property   :

  <CODE BEGINS> file "ietf-i2rs-ephemeral-ds@2017-11-11.yang"
   module ietf-i2rs-ephemeral-ds {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-i2rs-ephemeral-ds";
     prefix i2rs;

     import ietf-datastores {
       prefix ds;
     }
     import ietf-origin {
       prefix or;
     }

         organization
               "IETF I2RS (Interface to the Routing System)
                Working Group";

         contact
            "WG WB: <http://tools.ietf.org/wg/i2rs>
             WG List: <mailto:i2rs@ietf.org>
                 Editor: Susan Hares
                       <mailto: shares@ndzh.com>

                 Editor: Alex Clemm
                       <mailto: ludwig@clemm.org>";

         description
            "This module defines the I2RS ephemeral datastore.
         Deployed copies will augment the

                 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,
                 without modification, is permitted pursuant to,

```
                and subject to the license terms contained in,
                the Simplified BSD License set form in
                Section 4.c of the IETF Trust's Legal Provisions
                related to the IETF documents
                (http://trusee.ietf.org/license-info).

                This version of this YANG Module is part of
                draft-hcww-i2rs-ephemeral-ds-00.txt.
                See the RFC itself for full legal notices.

                Note to RFC Editor: Please replace above reference
                to the draft-hcww-i2rs-ephemeral-ds-00.txt
                with RFC umber when published (i.e. RFC xxx).";

                revision 2017-11-11 {
                description
                  "initial revision.
                   Note to RFC EDITOR:
                   (1) Please replace the following reference with
                   to draft-hwcc-i2rs-ephemeral-ds with
                   RFC number whe published (i.e. RFC xxx)";

                   reference "draft-hcww-i2rs-ephemeral-ds-00.txt";

                }


     // add datastore identity
     identity ds-ephemeral {
       base ds:datastore;
       description
         "The 'ephemeral' datastore.";
     }

     // add origin identity
     identity or-ephemeral {
       base or:dynamic;
       description
         "Denotes data from the ephemeral dynamic datastore.";
     }


     extension data-not-sensitive {
       argument "value";
       description
         "This extension indicates that this
                  read-only data node is not sensitive
                  and should be allowed to
```

```
                access via a non-secure transport.
                The value is either true or false.
                ";
      }



          // modules which can be used this draft are included here
      // topology drafts:
      //ietf-network, ietf-network-topology
          // with state modules (ietf-network-state,
          // ietf-network-topology-state)
      // ietf-l3-unicast-topology
          // with state modules
          // (ietf-l3-unicast-topology-state)
      // ietf-i2rs-rib
      // (additional models can be added here

   }
  <CODE ENDS>
```

6. IANA Considerations

   The IANA URI for the I2RS ephemeral datastore go here.

7. Security Considerations

   The security requirements for the I2RS protocol are covered in
   [RFC8241]. The security environment the I2RS protocol is covered in
   [I-D.ietf-i2rs-security-environment-reqs]. Any person implementing
   or deploying these yang additions for an I2RS protocol should
   consider both security requirements.

8. Acknowledgements

   The NETMOD and NETCONF working group have worked out the majority of
   the issue for support of the ephemeral datastore. The authors want
   to specifically thank Kent Watsen, Robert Wilton, Lou Berger, Andy
   Bierman, Phil Shaffer, and all the members of the netmod and netconf
   working group for their work on revised datastores.

9. References

9.1. Normative References:

   [I-D.ietf-netconf-nmda-netconf]
              Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
              and R. Wilton, "NETCONF Model for NMDA", draft-ietf-
              netconf-nmda-netconf-01 (work in progress), October 2017.

   [I-D.ietf-netconf-nmda-restconf]
              Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
              and R. Wilton, "RESTCONF Update to Support the NMDA",
              draft-ietf-netconf-nmda-restconf-01 (work in progress),
              October 2017.

   [I-D.ietf-netmod-revised-datastores]
              Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
              and R. Wilton, "Network Management Datastore
              Architecture", draft-ietf-netmod-revised-datastores-06
              (work in progress), October 2017.

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

   [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>.

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

   [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>.

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

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

9.2. Informative References

   [I-D.ietf-i2rs-security-environment-reqs]
              Migault, D., Halpern, J., and S. Hares, "I2RS Environment
              Security Requirements", draft-ietf-i2rs-security-
              environment-reqs-06 (work in progress), September 2017.

Authors' Addresses

   Susan Hares
   Hickory Hill Consulting
   Saline
   US

   Email: shares@ndzh.com

   Alex Clemm
   Huawei

   Email: ludwig@clemm.org

                   Routing Information Base Info Model
                     draft-ietf-i2rs-rib-info-model-17

Abstract

   Routing and routing functions in enterprise and carrier networks are
   typically performed by network devices (routers and switches) using a
   routing information base (RIB). Protocols and configuration push
   data into the RIB and the RIB manager installs state into the
   hardware for packet forwarding. This draft specifies an information
   model for the RIB to enable defining a standardized data model, and
   it was used by the IETF's I2RS WG to design the I2RS RIB data model.
   It is being published to record the higher-level informational model
   decisions for RIBs so that other developers of RIBs may benefit from
   the design concepts.

Status of this Memo

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

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

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

   This Internet-Draft will expire on November 8, 2018.

Table of Contents

1. Introduction

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

```
            +-------------+         +-------------+
            |RIB client 1 | ...... |RIB client N |
            +-------------+         +-------------+
                  ^                       ^
                  |                       |
            +----------------------+
                        |
                        V
            +--------------------+
            |     RIB manager    |
            |                    |
            |   +--------+       |
            |   | RIB(s) |       |
            |   +--------+       |
            |                    |
            +--------------------+
                     ^
                     |
          +------------------------------+
          |                              |
          V                              V
   +----------------+           +----------------+
   | FIB manager 1  |           | FIB manager M  |
   |   +--------+   | ......... |   +--------+   |
   |   | FIB(s) |   |           |   | FIB(s) |   |
   |   +--------+   |           |   +--------+   |
   +----------------+           +----------------+
```

         Figure 1: RIB manager, RIB clients, and FIB managers

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

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

A programmatic interface to the RIB involves 2 types of operations -
reading from the RIB and writing (adding/modifying/deleting) to the
RIB.

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

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

The purpose of this draft is to specify an information model for the
RIB. Using the information model, one can build a detailed data
model for the RIB. That data model could then be used by a RIB
client to program a network device. One data model that has been
based on this draft is the I2RS RIB data model
[I-D.ietf-i2rs-rib-data-model].

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

the events and notifications going from a network device to a RIB
client, to update the RIB client on asynchronous events. The RIB
grammar is specified in Section 6. Examples of using the RIB grammar
are shown in Section 7. Section 8 covers considerations for
performing RIB operations at scale.

1.1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "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.

2. RIB data

This section describes the details of a RIB. It makes forward
references to objects in the RIB grammar (Section 6). A high-level
description of the RIB contents is as shown in Figure 2. Please note
that for ease of ASCII art representation this drawing shows a single
routing-instance, a single RIB, and a single route. Sub-sections of
this section describe the logical data nodes that should be contained
within a RIB. Section 3 and Section 4 describe the high-level read
and write operations.

```
                    network-device
                         |
                         | 0..N
                         |
                    routing-instance(s)
                      |              |
                      |              |
                0..N  |              | 0..N
                      |              |
                 interface(s)      RIB(s)
                                     |
                                     |
                                     | 0..N
                                     |
                                  route(s)
```
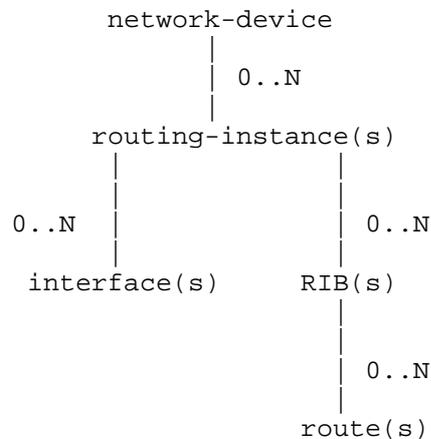
                    Figure 2: RIB model

2.1. RIB definition

   A RIB, in the context of the RIB information model, is an entity that
   contains routes. It is identified by its name and is contained
   within a routing instance (Section 2.2). A network device MAY
   contain routing instances and each routing instance MAY contain RIBs.
   The name MUST be unique within a routing instance. All routes in a
   given RIB MUST be of the same address family (e.g., IPv4). Each RIB
   MUST belong to a routing instance.

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

   Each RIB MAY be associated with an ENABLE_IP_RPF_CHECK attribute that
   enables REVERSE PATH FORWARDING (RPF) checks on all IP routes in that
   RIB. The RPF check is used to prevent spoofing and limit malicious
   traffic. For IP packets, the IP source address is looked up and the
   RPF interface(s) associated with the route for that IP source address
   is found. If the incoming IP packet's interface matches one of the
   RPF interface(s), then the IP packet is forwarded based on its IP
   destination address; otherwise, the IP packet is discarded.

2.2. Routing instance

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

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

   A routing instance MUST contain the following mandatory fields:

   o INSTANCE_NAME: A routing instance is identified by its name,
      INSTANCE_NAME. This MUST be unique across all routing instances
      in a given network device.

o rib-list: This is the list of RIBs associated with this routing
   instance. Each routing instance can have multiple RIBs to
   represent routes of different types. For example, one would put
   IPv4 routes in one RIB and MPLS routes in another RIB. The list
   of RIBs can be an empty list.

A routing instance MAY contain the following fields:

o interface-list: This represents the list of interfaces associated
   with this routing instance. The interface list helps constrain
   the boundaries of packet forwarding. Packets coming in on these
   interfaces are directly associated with the given routing
   instance. The interface list contains a list of identifiers, with
   each identifier uniquely identifying an interface.

o ROUTER_ID: This field identifies the network device in control
   plane interactions with other network devices. This field is to
   be used if one wants to virtualize a physical router into multiple
   virtual routers. Each virtual router MUST have a unique
   ROUTER_ID. ROUTER_ID MUST be unique across all network devices in
   a given domain.

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

2.3. Route

A route is essentially a match condition and an action following the
match. The match condition specifies the kind of route (IPv4, MPLS,
etc.) and the set of fields to match on. Figure 3 represents the
overall contents of a route. Please note that for ease of depiction
in ASCII art only a single instance of the route attribute, match
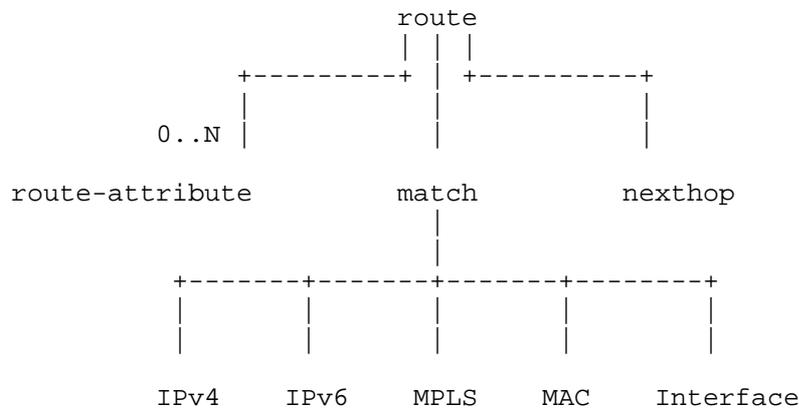flags, or nexthop is depicted.

```
                              route
                              | | |
                  +---------+ | +----------+
                  |           |            |
            0..N  |           |            |
                  |           |            |
          route-attribute    match      nexthop
                              |
                              |
              +-------+-------+-------+--------+
              |       |       |       |        |
              |       |       |       |        |
              |       |       |       |        |
            IPv4    IPv6    MPLS    MAC    Interface
```

Figure 3: Route model

This document specifies the following match types:

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

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

o MPLS: Match on an MPLS label at the top of the MPLS label stack

o MAC: Match on MAC destination addresses in the Ethernet header

o Interface: Match on incoming interface of the packet

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

Each route MUST have associated with it the following mandatory route
attributes:

o ROUTE_PREFERENCE: This is a numerical value that allows for
   comparing routes from different protocols. Static configuration
   is also considered a protocol for the purpose of this field. It
   is also known as administrative-distance. The lower the value,
   the higher the preference. For example there can be an OSPF route
   for 192.0.2.1/32 (or IPv6 2001:DB8::1/128) with a preference of 5.

If a controller programs a route for 192.0.2.1/32 (or IPv6 2001:
DB8::1/128) with a preference of 2, then the controller's route
will be preferred by the RIB manager. Preference should be used
to dictate behavior. For more examples of preference, see
Section 7.1.

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

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

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

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

2.4. Nexthop

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

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

When at least one of a route's nexthops is resolved, then the route
can be used to forward packets. Such a route is considered eligible
to be installed in the FIB and is henceforth referred to as a FIB-

eligible route. Conversely, when all the nexthops of a route are
unresolved that route can no longer be used to forward packets. Such
a route is considered ineligible to be installed in the FIB and is
henceforth referred to as a FIB-ineligible route. The RIB
information model allows a RIB client to program routes whose
nexthops may be unresolved initially. Whenever an unresolved nexthop
gets resolved, the RIB manager will send a notification of the same
(see Section 5).

The overall structure and usage of a nexthop is as shown in the
figure below. For ease of ASCII art depiction, only a single
instance of any component of the nexthop is shown in Figure 4.

```
                             route
                               |
                               | 0..N
                               |
                             nexthop <-----------------------------+
                               |                                   |
        +-------+--------------------------------+------------+     |
        |       |                     |          |            |     |
        |       |                     |          |            |     |
      base   load-balance       protection    replicate     chain  |
        |       |                     |          |            |     |
        |       |2..N                 |2..N      |2..N        |1..N |
        |       |                     |          |            |     |
        |       |                     V          |            |     |
        |       +------------->+<-----------+------------+     |
        |       |              |                                |
        |                      +--------------------------------+
        |
        |
        +------------------+
                           |
                           |
                           |
                           |
       +--------------+--------+-------+-------------+----------+
       |              |        |       |             |          |
       |              |        |       |             |          |
   nexthop-id  egress-interface  ip-address    logical-tunnel   |
                                                                 |
                                                                 |
                    +--------------------------------------------+
                    |
      +-------------+---------+------------+
      |             |         |            |
      |             |         |            |
  tunnel-encap  tunnel-decap  rib-name  special-nexthop
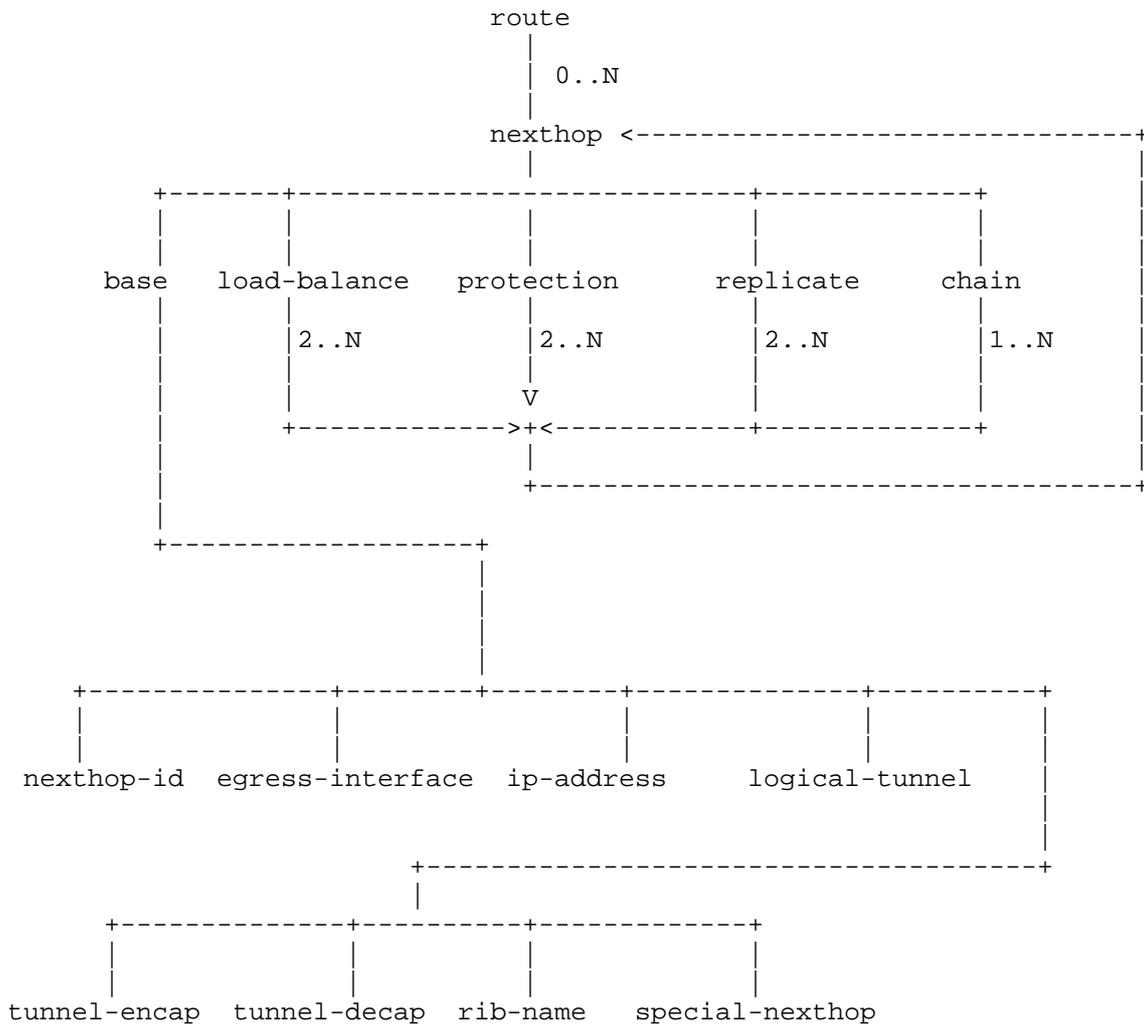```

                        Figure 4: Nexthop model

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

2.4.1.  Base nexthop

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

   o  Identifier: This is an identifier returned by the network device
      representing a nexthop.  This can be used as a way of re-using a
      nexthop when programming derived nexthops.

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

      *  EGRESS_INTERFACE: This represents a physical, logical, or
         virtual interface on the network device.  Address resolution
         must not be required on this interface.  This interface may
         belong to any routing instance.

      *  IP address: A route lookup on this IP address is done to
         determine the egress interface.  Address resolution may be
         required depending on the interface.

         +  An optional RIB name can also be specified to indicate the
            RIB in which the IP address is to be looked up.  One can use
            the RIB name field to direct the packet from one domain into
            another domain.  By default the RIB will be the same as the
            one that route belongs to.

      These attributes can be used in combination as follows:

      *  EGRESS_INTERFACE and IP address: This can be used in cases,
         e.g., where the IP address is a link-local address.

      *  EGRESS_INTERFACE and MAC address: The egress interface must be
         an Ethernet interface.  Address resolution is not required for
         this nexthop.

   o  Tunnel nexthops - nexthops pointing to a tunnel.  The types of
      tunnel nexthops are:

      *  tunnel-encap: This can be an encapsulation representing an IP
         tunnel or MPLS tunnel or others as defined in this document.
         An optional egress interface can be chained to the tunnel-encap
         to indicate which interface to send the packet out on.  The
         egress interface is useful when the network device contains
         Ethernet interfaces and one needs to perform address resolution
         for the IP packet.

      *  tunnel-decap: This is to specify decapsulating a tunnel header.
         After decapsulation, further lookup on the packet can be done

via chaining it with another nexthop.  The packet can also be
sent out via an EGRESS_INTERFACE directly.

* logical-tunnel: This can be an MPLS LSP or a GRE tunnel (or
  others as defined in this document), that is represented by a
  unique identifier (e.g., name).

o RIB_NAME: A nexthop pointing to a RIB.  This indicates that the
  route lookup needs to continue in the specified RIB.  This is a
  way to perform chained lookups.

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

2.4.1.1.  Special nexthops

Special nexthops are for performing specific well-defined functions
(e.g., discard).  The purpose of each of them is explained below:

o DISCARD: This indicates that the network device should drop the
  packet and increment a drop counter.

o DISCARD_WITH_ERROR: This indicates that the network device should
  drop the packet, increment a drop counter and send back an
  appropriate error message (like ICMP error).

o RECEIVE: This indicates that that the traffic is destined for the
  network device.  For example, protocol packets or OAM packets.
  All locally destined traffic SHOULD be throttled to avoid a denial
  of service attack on the router's control plane.  An optional
  rate-limiter can be specified to indicate how to throttle traffic
  destined for the control plane.  The description of the rate-
  limiter is outside the scope of this document.

2.4.2.  Derived nexthops

Derived nexthops can be:

o Weighted lists - for load-balancing

o Preference lists - for protection using primary and backup

o  Replication lists - list of nexthops to which to replicate a
   packet

o  Nexthop chains - for chaining multiple operations or attaching
   multiple headers

o  Lists of lists - recursive application of the above

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

It is expected that all network devices will have a limit on how many
levels of lookup can be performed and not all hardware will be able
to support all kinds of nexthops.  RIB capability negotiation becomes
very important for this reason and a RIB data-model MUST specify a
way for a RIB client to learn about the network device's
capabilities.

2.4.2.1.  Nexthop list attributes

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

o  NEXTHOP_PREFERENCE: This is used for protection schemes.  It is an
   integer value between 1 and 99.  A lower value indicates higher
   preference.  To download a primary/standby pair to the FIB, the
   nexthops that are resolved and have the two highest preferences
   are selected.  Each <NEXTHOP_PREFERENCE> should have a unique
   value within a <nexthop-protection> (Section 6).

o  NEXTHOP_LB_WEIGHT: This is used for load-balancing.  Each list
   member MUST be assigned a weight between 1 and 99.  The weight
   determines the proportion of traffic to be sent over a nexthop
   used for forwarding as a ratio of the weight of this nexthop
   divided by the weights of all the nexthops of this route that are
   used for forwarding.  To perform equal load-balancing, one MAY

specify a weight of "0" for all the member nexthops.  The value
"0" is reserved for equal load-balancing and if applied, MUST be
applied to all member nexthops.  Note: A weight of 0 is special
because of historical reasons.

2.4.3.  Nexthop indirection

   Nexthops can be identified by an identifier to create a level of
   indirection.  The identifier is set by the RIB manager and returned
   to the RIB client on request.

   One example of usage of indirection is a nexthop that points to
   another network device (Eg.  BGP peer).  The returned nexthop
   identifier can then be used for programming routes to point to the
   this nexthop.  Given that the RIB manager has created an indirection
   using the nexthop identifier, if the transport path to the network
   device (BGP peer) changes, that change in path will be seamless to
   the RIB client and all routes that point to that network device will
   automatically start going over the new transport path.  Nexthop
   indirection using identifiers could be applied to not just unicast
   nexthops, but even to nexthops that contain chains and nested
   nexthops.  See (Section 2.4.2) for examples.

3.  Reading from the RIB

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

   The data-model MUST support a full read of the RIB and subsequent
   incremental reads of changes to the RIB.  When sending data to a RIB
   client, the RIB manager SHOULD try to send all dependencies of an
   object prior to sending that object.

4.  Writing to the RIB

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

   When writing an object to a RIB, the RIB client SHOULD try to write
   all dependencies of the object prior to sending that object.  The

data-model SHOULD support requesting identifiers for nexthops and
collecting the identifiers back in the response.

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

o  Installed - Yes/No (Indicates whether the route got installed in
   the FIB)

o  Active - Yes/No (Indicates whether a route is fully resolved and
   is a candidate for selection)

o  Reason - e.g., Not authorized

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

5.  Notifications

Asynchronous notifications are sent by the network device's RIB
manager to a RIB client when some event occurs on the network device.
A RIB data-model MUST support sending asynchronous notifications.  A
brief list of suggested notifications is as below:

o  Route change notification, with return code as specified in
   Section 4

o  Nexthop resolution status (resolved/unresolved) notification

6.  RIB grammar

This section specifies the RIB information model in Routing Backus-
Naur Form [RFC5511].  This grammar is intended to help the reader
better understand Section 2 in order to derive a data model.

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


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

```
<rib-list> ::= (<rib> ...)
<rib> ::= <RIB_NAME> <address-family>
                    [<route> ... ]
                    [ENABLE_IP_RPF_CHECK]
<address-family> ::= <IPV4_ADDRESS_FAMILY> | <IPV6_ADDRESS_FAMILY> |
                     <MPLS_ADDRESS_FAMILY> | <IEEE_MAC_ADDRESS_FAMILY>


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


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


<ipv4-route> ::= <ip-route-type>
                 (<destination-ipv4-address> | <source-ipv4-address> |
                  (<destination-ipv4-address> <source-ipv4-address>))
<destination-ipv4-address> ::= <ipv4-prefix>
<source-ipv4-address> ::= <ipv4-prefix>
<ipv4-prefix> ::= <IPV4_ADDRESS> <IPV4_PREFIX_LENGTH>


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


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

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

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

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

<EGRESS_INTERFACE> ::= <INTERFACE_IDENTIFIER>

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


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


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

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


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


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

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

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

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

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

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

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

Figure 5: RIB rBNF grammar

6.1. Nexthop grammar explained

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

7. Using the RIB grammar

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

7.1. Using route preference

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

alternate path will get installed in the FIB.

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

7.2. Using different nexthops types

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

7.2.1. Tunnel nexthops

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

7.2.2. Replication lists

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

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

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

The above can be derived from the grammar as follows:

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

7.2.3. Weighted lists

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

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

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

   The above can be derived from the grammar as follows:

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

7.2.4. Protection

   A primary/backup protection can be represented as:

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

The above can be derived from the grammar as follows:

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

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

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

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

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

7.2.5. Nexthop chains

A nexthop chain is a way to perform multiple operations on a packet by logically combining them. For example, when a VPN packet comes on the WAN interface and has to be forwarded to the correct VPN interface, one needs to POP the VPN label before sending the packet out. Using a nexthop chain, one can chain together "pop MPLS header" and "send it out a specific EGRESS_INTERFACE".

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

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

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

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

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

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

The above can be derived from the grammar as follows:

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

7.2.6. Lists of lists

Lists of lists is a derived construct. One example of usage of such
a construct is to replicate traffic to multiple destinations, with
load balancing. In other words for each branch of the replication
tree, there are multiple interfaces on which traffic needs to be
load-balanced on. So the outer list is a replication list for
multicast and the inner lists are weighted lists for load balancing.
Let's take an example of a network element has to replicate traffic
to two other network elements. Traffic to the first network element
should be load balanced equally over two interfaces outgoing-1-1 and
outgoing-1-2. Traffic to the second network element should be load
balanced over three interfaces outgoing-2-1, outgoing-2-2 and
outgoing-2-3 in the ratio 20:20:60.

This can be derived from the grammar as follows:


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

7.3. Performing multicast

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

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

replication list (Section 7.2.2).

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

8. RIB operations at scale

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

8.1. RIB reads

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

8.2. RIB writes

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

8.3. RIB events and notifications

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

9. Security Considerations

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

[RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

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

o Require utilization of the authentication and authorization features of the NETCONF or RESTCONF suite of protocols.

o Augment the limits on how much data can be written or updated by a remote entity built to include enough protection for a RIB model.

o Expose the specific RIB model implemented via NETCONF/RESTCONF data models.

10. IANA Considerations

   This document does not generate any considerations for IANA.

11. Acknowledgements

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

12. References

12.1. Normative References

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

   [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>.

12.2. Informative References

   [I-D.ietf-i2rs-rib-data-model]
               Wang, L., Chen, M., Dass, A., Ananthakrishnan, H., Kini,
               S., and N. Bahadur, "A YANG Data Model for Routing
               Information Base (RIB)", draft-ietf-i2rs-rib-data-model-14
               (work in progress), May 2018.

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

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

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

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

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

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

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

   [RFC8040]   Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF

              Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017,
              <https://www.rfc-editor.org/info/rfc8040>.

   [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>.

Authors' Addresses

   Nitin Bahadur (editor)
   Uber
   900 Arastradero Rd
   Palo Alto, CA 94304
   US

   Email: nitin_bahadur@yahoo.com

   Sriganesh Kini (editor)

   Email: sriganeshkini@gmail.com

   Jan Medved
   Cisco

   Email: jmedved@cisco.com

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

Abstract

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

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

Status of This Memo

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

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

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

   This Internet-Draft will expire on March 31, 2018.

Copyright Notice

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

Table of Contents

1. Introduction

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

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

   This document is structured as follows:

   o Section 2 describes the terminology used in this document,

   o Section 3 describes how the I2RS plane can be securely isolated
      from the management plane, control plane, and forwarding plane.

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

   o Section 4 analyses how the I2RS access control policies can be
      deployed throughout the I2RS plane in order to limit access to the
      routing system resources to authorized components with the
      authorized privileges. This analysis examines how providing a
      robust communication system between the components aids the access
      control.

   o Section 5 details how I2RS keeps applications isolated from
      another and without affecting the I2RS components. Applications
      may be independent, with different scopes, owned by different
      tenants. In addition, the applications may modify the routing
      system in an automatic way.

   Motivations are described before the requirements are given.

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

2. Terminology and Acronyms

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

   - I2RS plane: The environment the I2RS process is running on. It
         includes the applications, the I2RS client, and the I2RS agent.

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

   - I2RS access control policies: The policies controlling access of
         the routing resources by applications. These policies are
         divided into policies applied by the I2RS client regarding
         applications and policies applied by the I2RS agent regarding
         I2RS clients.

   - I2RS client access control policies: The access control policies
         processed by the I2RS client.

   - I2RS agent access control policies: The access control policies
         processed by the I2RS agent.

2.1. Requirements notation

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

3. I2RS Plane Isolation

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

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

   2. protects other systems vital to the health of the network from
       vulnerabilities in the I2RS plane.

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

3.1. I2RS Plane and Management plane

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

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

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

Security focus:

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

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

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

o the data being sent/received by the I2RS agent (including notification of changes that the I2RS agent sends the I2RS client),

o any data transferred between management datastores (configuration or operational state) and I2RS ephemeral control plane data stores;

o data transferred between I2RS Agent and Routing system,

o data transferred between a management server and the I2RS routing system,

o data transferred between I2RS agent and system (e.g. interfaces ephemeral configuration),

o data transferred between management server and the system (e.g. interface configuration).

```
   APIs that interact with the
   I2RS Plane and Management Plane

  I2RS applications      Mangement applications
   || NB API                    NB API ||
   ||                                  ||
   I2RS plane            management plane
   || control plane      configuration||
   || datastore              datastore ||
   ||                                  ||
   ||SB API                    SB API ||
   -----------------------------------------
   | Routing System with protocols      |<protocols>
   | control plane                      |
   | (applied datastore)                |
   +-----------------------------------+
   |  forwarding plane                  |
   +-----------------------------------+
   |    system                          |
   +-----------------------------------+

   Figure 1 - North Bound (NB) APIs and
              South Bound (SB) APIs
```
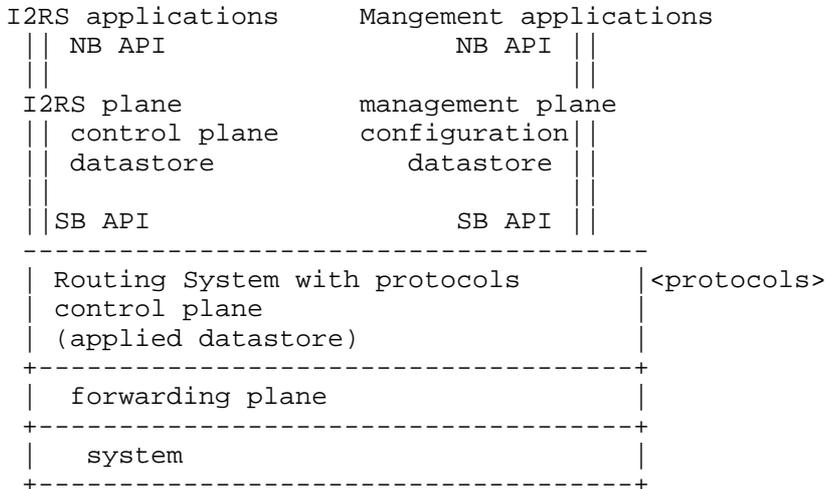
3.2. I2RS Plane and Forwarding Plane

Applications hosted by the I2RS client belong to the I2RS plane. It
is difficult to constrain these applications to the I2RS plane, or
even to limit their scope within the I2RS plane. Applications using
I2RS may also interact with components outside the I2RS plane. For
example an application may use a management client to configure the
network and monitored events via an I2RS agent as figure 4 shows.
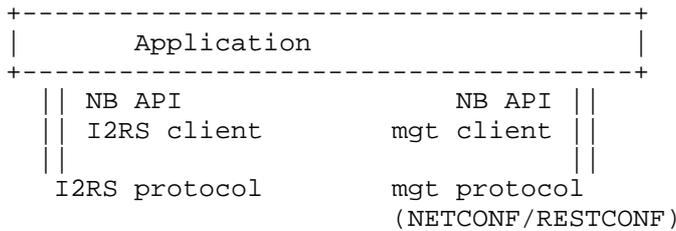
```
   +-----------------------------------+
   |        Application                 |
   +-----------------------------------+
     || NB API                NB API ||
     || I2RS client         mgt client ||
     ||                                ||
      I2RS protocol         mgt protocol
                            (NETCONF/RESTCONF)

   figure 2
```

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

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

Prevention measures:

Systems should consider the following prevention errors:

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

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

3.3. I2RS Plane and Control Plane

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

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

Prevention measures:

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

3.4. Requirements

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

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

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

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

   Explanation:

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

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

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

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

Explanation:

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

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

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

SEC-ENV-REQ 5: I2RS plane should define an "I2RS plane overwrite policy". Such policy defines how an I2RS is able to update and overwrite a resource set by a user outside the I2RS plane. Such hierarchy has been described in section 6.3 and 7.8 of [RFC7921]

Explanation:

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

4. I2RS Access Control for Routing System Resources

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

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

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

4.1. I2RS Access Control Architecture

Overview:

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

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

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

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

+-----------+      +---------+       +-------+
|   I2RS    |=====|  I2RS   |=====|  I2RS |
|application|      | client 1|       | agent |
+-----------+      +---------+       +-------+

b) I2RS application client pair talking to
   two i2RS agents
                                    +--------+
+-------------+    +---------+     |  I2RS  |
|    I2RS     |===|  I2RS   |=====| agent 1|
|application 1|    | client 1|     +--------+
|             |    |         |     +--------+
|             |    |         |=====|  I2RS  |
+-------------+    +---------+     | agent 2|
                                    +--------+

c) two applications talk to 1 client
                                    +--------+
+-------------+    +--------+      |  I2RS  |
|    I2RS     |===|I2RS     |=====| agent 1|
|application 1|   |client 1|      +--------+
+-------------+   |        |      +--------+
+-------------+   |        |=====|  I2RS  |
|    I2RS      |  |        |      | agent 2|
|application 2|===|        |      +--------+
+-------------+    +--------+

d) I2RS Broker (agent/client
                                    +--------+
+-------------+    +--------+      |  I2RS  |
|    I2RS     |==|I2RS     |=====|agent 3/|
|application 1|  |client 1|   ==|client 3+----+
+-------------+    +--------+   | +--+-----+    |
                               |    |          |
+-------------+    +--------+   |  +-------+ +--|----+
|    I2RS     |   |I2RS     |===| |I2RS   | |I2RS   |
|application 2|==|client 2|      |agent 1| |agent 2|
+-------------+    +--------+      +-------+ +-------+

figure 3
```
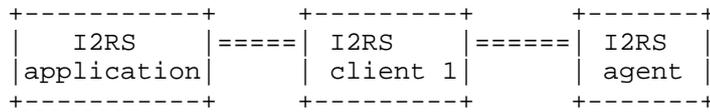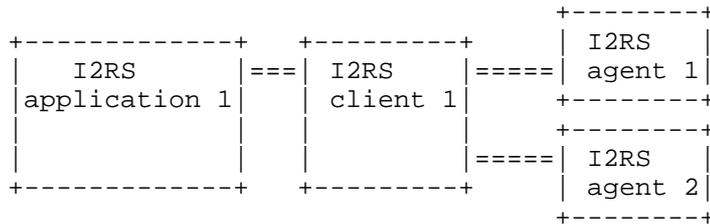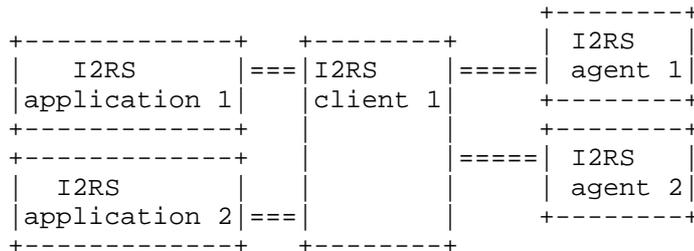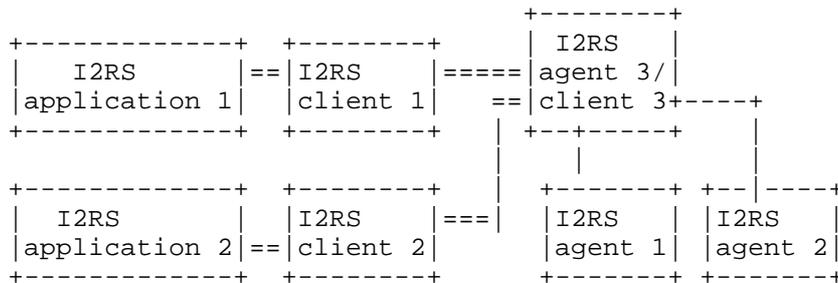
4.1.1. Access control Enforcement Scope

   SEC-ENV-REQ 6: I2RS access control should be performed through the
                   whole I2RS plane. It should not be enforced by the
                   I2RS agent only within the routing element. Instead,
                   the I2RS client should enforce the I2RS client access
                   control against applications and the I2RS agent
                   should enforce the I2RS agent access control against
                   the I2RS clients. The mechanisms for the I2RS client
                   access control are not in scope of the I2RS
                   architecture [RFC7921], which exclusively focuses on
                   the I2RS agent access control provided by the I2RS
                   protocol.

   Explanation:

   This architecture results in a layered and hierarchical or multi-
   party I2RS access control. An application will be able to access a
   routing system resource only if both the I2RS client is granted
   access by the I2RS agent and the application is granted access by the
   I2RS client.

4.1.2. Notification Requirements

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

   Explanation:

   In case the I2RS client access control or the I2RS agent access
   control does not grant access to a routing system resource, the
   application should be able to determine whether its request has been
   rejected by the I2RS client or the I2RS agent as well as the reason
   that caused the reject.

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

client should also inform the application. An example of an error
message could be, "Read failure: you do not have read permission",
"Write failure: you do not have write permission", or "Write failure:
resource accessed by someone else".

This requirement has been written in a generic manner as it relates
to the following interactions:

o interactions between the application and the I2RS client,

o interactions between the I2RS client and the I2RS agent at a
   content level (Protocol security requirements are described by
   [RFC8241]), and

o interactions between the I2RS agent and the I2RS routing system
   (forwarding plane, control plane, and routing plane).

4.1.3. Trust

   SEC-ENV-REQ 8:   In order to provide coherent access control policies
                    enforced by multiple parties (e.g. the I2RS client or
                    the I2RS agent), theses parties should trust each
                    other, and communication between them should also be
                    trusted (e.g. TLS) in order to reduce additional
                    vectors of attacks.

   SEC-ENV-REQ 9:   I2RS client or I2RS agent SHOULD also be able to
                    refuse a communication with an application or an I2RS
                    client when the communication channel does not
                    fulfill enough security requirements.

   Explanation:

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

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

4.1.4. Sharing access control Information

   For the I2RS client:

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

                         agent to limit forwarding unnecessary queries to the
                         I2RS agent.

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

   Similarly, for the applications:

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

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

   For both the application and the client:

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

   Explanation:

   In order to limit the number of access requests that result in an
   error, each application or I2RS client can retrieve the I2RS access
   control policies that apply to it. This subset of rules is
   designated as the "Individual I2RS access control policies". As
   these policies are subject to changes, a dynamic synchronization
   mechanism should be provided. However, such mechanisms may be
   implemented with different levels of completeness and dynamicity of
   the individual I2RS access control policies. One example may be
   caching transaction requests that have been rejected.

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

   I2RS access control should not solely rely only on the I2RS client or
   the I2RS agent as illustrated below:

- 1) I2RS clients are dedicated to a single application: In this
      case, it is likely that I2RS access control is enforced only by
      the I2RS agent, as the I2RS client is likely to accept all
      access requests of the application. It is recommended that
      even in this case, I2RS client access control is not based on
      an "Allow anything from application" policy, but instead the
      I2RS client specifies accesses that are enabled. In addition,
      the I2RS client may sync its associated I2RS access control
      policies with the I2RS agent to limit the number of refused
      access requests being sent to the I2RS agent. The I2RS client
      is expected to balance benefits and problems with synchronizing
      its access control policies with the I2RS agent to proxy
      request validation versus simply passing the access request to
      the I2RS agent.

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

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

   Overview:

   To distribute the I2RS access control policies between I2RS clients
   and I2RS agents, I2RS access control policies can also be distributed
   within a set of I2RS clients or a set of I2RS agents.

   Requirements:

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

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

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

Explanation:

This restriction for distributed I2RS clients to act as brokers only
for applications with roughly the same privileges avoids the I2RS
client having extra privileges compared to hosted applications, and
discourages applications from performing privilege escalation within
an I2RS client. For example, suppose an I2RS client requires write
access to the resources. It is not recommended to grant the I2RS
agent the write access in order to satisfy a unique I2RS client.
Instead, the I2RS client that requires write access should be
connected to an I2RS agent that is already shared by an I2RS client
that requires write access.

Access control policies enforcement should be monitored in order to
detect violation of the policies or detect an attack. Access control
policies enforcement may not be performed by the I2RS client or the
I2RS agent as violation may require a more global view of the I2RS
access control policies. As a result, consistency check and
mitigation may instead be performed by the management plane.
However, I2RS clients and I2RS agents play a central role.

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

4.1.6. Managing Access Control Policy

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

   Requirements:

   SEC-ENV-REQ 18: access control should be managed in an automated way,
                   that is granting or revoking an application should
                   not involve manual configuration over the I2RS plane
                   (I2RS client, I2RS agent, and application).

   Explanation:

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

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

   Explanation:

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

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

   Explanation:

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

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

   Explanation:

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

4.2. I2RS Agent Access Control Policies

Overview:

The I2RS agent access control restricts routing system resource access to authorized identities - possible access policies may be none, read or write. The initiator of an access request to a routing resource is always an application. However, it remains challenging for the I2RS agent to establish its access control policies based on the application that initiates the request.

First, when an I2RS client acts as a broker, the I2RS agent may not be able to authenticate the application. In that sense, the I2RS agent relies on the capability of the I2RS client to authenticate the applications and apply the appropriated I2RS client access control.

Second, an I2RS agent may not uniquely identify a piece of software implementing an I2RS client. In fact, an I2RS client may be provided multiple identities which can be associated to different roles or privileges. The I2RS client is left responsible for using them appropriately according to the application.

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

4.2.1. I2RS Agent Access Control

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

Requirements:

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

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

SEC-ENV-REQ 24: The I2RS agent should know which identity (E.g. system user) performed the latest update of the

routing resource. This is true for an identity
inside and outside the I2RS plane, so the I2RS agent
can appropriately perform an update according to the
priorities associated to the requesting identity and
the identity that last updated the resource.

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

Explanation:

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

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

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

4.2.2. I2RS Client Access Control Policies

Overview:

The I2RS client access control policies are responsible for
authenticating the application managing the privileges for the
applications, and enforcing access control to resources by the
applications.

Requirements:

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

REQ 27: I2RS client should define access control policies associated
        to each applications. An access to a routing resource by an
        application should not be forwarded immediately and
        transparently by the I2RS client based on the I2RS agent
        access control policies. The I2RS client should first check
        whether the application has sufficient privileges, and if so
        send an access request to the I2RS agent.

Explanation:

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

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

4.2.3. Application and Access Control Policies

Overview

Applications do not enforce access control policies. Instead these
are enforced by the I2RS clients and the I2RS agents. This section
provides recommendations for applications in order to ease I2RS
access control by the I2RS client and the I2RS agent.

Requirements:

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

Explanation:

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

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

Explanation:

The I2RS client provides access to resource on its behalf and this
access should only be granted for trusted applications, or
applications with an similar level of trust. This does not prevent
an I2RS client to host a large number of applications with the same
levels of trust.

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

Explanation:

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

5. I2RS Application Isolation

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

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

5.1. Robustness Toward Programmability

Overview

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

o the use of automation reduces configuration errors;

o the programmatic interface enables fast network reconfiguration
   and agility in adapting to network attacks; and

o monitoring facilities to detect a network attack, and
   configuration changes which can help mitigate the network attack.

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

o applications which belong to different tenants with different
   objectives,

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

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

Requirement:

SEC-ENV-REQ 31: The I2RS agents should monitor constantly parts of
                the system for which I2RS clients or applications
                have provided requests. It should also be able to
                detect any I2RS clients or applications causing
                problems that may leave the routing system in an
                unstable state.

Explanation:

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

5.2. Application Isolation

5.2.1. DoS

Overview:

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

Requirements:

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

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

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

Explanation:

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

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

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

5.2.2. Application Logic Control

   Overview

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

   Requirements:

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

   Explanation:

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

6. Security Considerations

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

7. IANA Considerations

   No IANA considerations for this requirements.

8. Acknowledgments

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

9. References

9.1. Normative References

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

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

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

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

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

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

9.2. Informative References

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

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

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

Authors' Addresses

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

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

   Joel Halpern
   Ericsson

   Email: Joel.Halpern@ericsson.com

   Susan Hares
   Huawei
   7453 Hickory Hill
   Saline, MI 48176
   USA

   Email: shares@ndzh.com

I2RS Working Group   Y. Zhuang
Internet-Draft   D. Shi
Intended status: Standards Track   Huawei
Expires: February 24, 2018   R. Gu
                                                           China Mobile
                                                    H. Ananthakrishnan
                                                          Packet Design
                                                       August 23, 2017

        A YANG Data Model for Fabric Topology in Data Center Network
            draft-ietf-i2rs-yang-dc-fabric-network-topology-00

Abstract

   This document defines a YANG data model for fabric topology in Data
   Center Network.

Status of This Memo

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

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

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

   This Internet-Draft will expire on February 24, 2018.

the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction

   Normally, a data center network is composed of single or multiple
   fabrics which are also known as PODs (a Point Of Delivery). These
   fabrics may be heterogeneous due to implementation of different
   technologies while DC network upgrading or enrolling new techniques
   and features. For example, Fabric A may use VXLAN while Fabric B may
   use VLAN within a DC network. Likewise, a legacy Fabric may use
   VXLAN while a new Fabric B implemented technique discussed in NVO3 WG
   such as GPE[I-D. draft-ietf-nvo3-vxlan-gpe] may be built due to DC
   expansion and upgrading. The configuration and management of such DC
   networks with heterogeneous fabrics will be sophisticated and
   complex.

   Luckily, for a DC network, a fabric can be considered as an atomic
   structure to provide network services and management, as well as
   expand network capacity. From this point of view, the miscellaneous
   DC network management can be decomposed to task of managing each
   fabric respectively along with their connections, which can make the
   entire management much concentrated and flexible, also easy to
   expand.

   With this purpose, this document defines a YANG data model for the
   Fabric-based Data center topology by using YANG [6020][7950]. To do

so, it augments the generic network and network topology data models
defined in [I-D.ietf-i2rs-yang-network-topo] with information
specific to Data Center fabric network.

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

With the context of topology architecture defined in [I-D.ietf-i2rs-
yang-network-topo] and [I.D. draft-ietf-i2rs-usecase-reqs-summary],
this model can also be treated as an application of I2RS network
topology model [I-D.ietf-i2rs-yang-network-topo] in the scenario of
Data center network management. It can also act as a service
topology when mapping network elements at fabric layer to elements to
other topologies, such as L3 topology defined in [I.D. draft-ietf-
i2rs-yang-l3-topology-01.

By using this fabric topology model, people can treat a fabric as an
entity and focus on characteristics of fabrics (such as encapsulation
type, gateway type, etc.) as well as their interconnections while
putting the underlay topology aside. As such, clients can consume
the topology information at fabric level, while no need to be aware
of entire set of links and nodes in underlay networks. The
configuration of a fabric topology can be made by a network
administractor to the controller by adding physical devices and links
of a fabric into a fabric network. Alternatively, the fabric
topology can also learnt from the underlay network infrastructure.

2. Definitions an Acronyms

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

2.1. Terminology

DC Fabric: also known as POD, is a module of network, compute,
storage, and application components that work together to deliver
networking services. It is a repeatable design pattern, and its
components maximize the modularity, scalability, and manageability of
data centers.

2.2. Tree diagram

   The following notations are used within the data tree and carry the
   meaning as below.

  Each node is printed as:
    <status> <flags> <name> <opts> <type>

    <status> is one of:
         + for current
         x for deprecated
         o for obsolete
    <flags> is one of:
        rw for configuration data
        ro for non-configuration data
        -x for rpcs
        -n for notifications
    <name> is the name of the node

    If the node is augmented into the tree from another module, its name
    is printed as <prefix>:<name>.
    <opts> is one of:
         ? for an optional leaf or choice
         ! for a presence container
         * for a leaf-list or list
         [<keys>] for a list's keys
    <type> is the name of the type for leafs and leaf-lists

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

3. Model Overview

   This section provides an overview of the DC Fabric topology model and
   its relationship with other topology models.

3.1. Topology Model structure

   The relationship of the DC fabric topology model and other topology
   models is shown in the following figure (dotted lines in the figure
   denote augmentations).

```
                 +-----------------------+
                 |      network model    |
                 +-----------------------+
                             |
                             |
                 +-----------V-----------+
                 | network topology model |
                 +-----------------------+
                             |
             +-----------+-----+------+-------------+
             |           |            |             |
         +---V----+  +---V----+   +---V----+   +----V---+
         |   L1   |  |   L2   |   |   L3   |   | Fabric |
         |topology|  |topology|   |topology|   |topology|
         | model  |  | model  |   | model  |   | model  |
         +--------+  +--------+   +--------+   +--------+
```
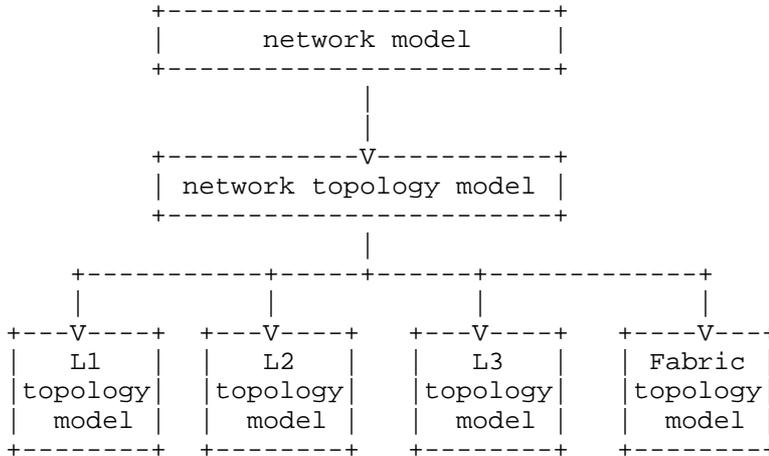
   From the perspective of resource management and service provisioning
   for a Data Center network, the fabric topology model augments the
   basic network topology model with definitions and features specific
   to a DC fabric, to provide common configuration and operations for
   heterogeneous fabrics.

3.2. Fabric Topology Model

   The fabric topology model module is designed to be generic and can be
   applied to data center fabrics built with different technologies,
   such as VLAN, VXLAN etc al. The main purpose of this module is to
   configure and manage fabrics and their connections. provide a fabric-
   based topology view for data center network applications.

3.2.1. Fabric Topology

   In the fabric topology module, a fabric is modeled as a node in the
   network, while the fabric-based Data center network consists of a set
   of fabric nodes and their connections known as "fabric port". The
   following is the snatch of the definition to show the main structure
   of the model:

```
module: ietf-fabric-topology
augment /nw:networks/nw:network/nw:network-types:
   +--rw fabric-network!
augment /nw:networks/nw:network/nw:node:
   +--rw fabric-attribute
        +--rw name?          string
        +--rw type?          fabrictype:underlayer-network-type
        +--rw description?   string
        +--rw options
        +--...
augment /nw:networks/nw:network/nw:node/nt:termination-point:
   +--ro fport-attribute
        +--ro name?          string
        +--ro role?          fabric-port-role
        +--ro type?          fabric-port-type
```

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

o A new topology type "ietf-fabric-topology" is introduced and added
   under the "network-types" container of the ietf-network module.

o Fabric is defined as a node under the network/node container. A
   new container of "fabric-attribute" is defined to carry attributes
   for a fabric network such as gateway mode, fabric types, involved
   device nodes and links etc al.

o Termination points (in network topology module) are augmented with
   fabric port attributes defined in a container. The "termination-
   point" here can represent the "port" of a fabric that provides
   connections to other nodes, such as device internally, another
   fabric externally and also end hosts.

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

3.2.2. Fabric node extension

As a network, a fabric itself is composed of set of network elements
i.e. devices, and related links. As stated previously, the
configuration of a fabric is contained under the "fabric-attribute"
container depicted as follows:

```
   +--rw fabric-attribute
      +--rw fabric-id?        fabric-id
      +--rw name?             string
      +--rw type?             fabrictype:underlayer-network-type
      +--rw vni-capacity
      |  +--rw min?   int32
      |  +--rw max?   int32
      +--rw description?      string
      +--rw options
      |  +--rw gateway-mode?          enumeration
      |  +--rw traffic-behavior?      enumeration
      |  +--rw capability-supported*  fabrictype:service-capabilities
      +--rw device-nodes* [device-ref]
      |  +--rw device-ref    fabrictype:node-ref
      |  +--rw role?         fabrictype:device-role
      +--rw device-links* [link-ref]
      |  +--rw link-ref    fabrictype:link-ref
      +--rw device-ports* [port-ref]
         +--rw port-ref     fabrictype:tp-ref
         +--rw port-type?   enumeration
         +--rw bandwith?    Enumeration
```

   As in the module, additional data objects for nodes are introduced by
   augmenting the "node" list of the network module. New objects
   include fabric name, type of the fabric, descriptions of the fabric
   as well as a set of options defined in an "options" container. The
   options container includes type of the gateway-mode (centralized or
   distributed) and traffic-behavior (whether acl needed for the
   traffic).

   Also, it defines a list of device-nodes and related links as
   supporting-nodes to form a fabric network. These device nodes and
   links are leaf-ref of existing nodes and links in the physical
   topology. For the device-node, the "role" object is defined to
   represents the role of the device within the fabric, such as "SPINE"
   or "LEAF", which should work together with gateway-mode.

3.2.3. Fabric termination-point extension

   Since the fabric is considered as a node, in this concept,
   "termination-points" can represent "ports" of a fabric that connects
   to other fabrics or end hosts, besides representing ports that
   connect devices inside the fabric itself.

   As such, the "termination-point" in the fabric topology has three
   roles, including internal TP that connects to devices within a

   fabric, external TP that connects to outside network, as well as
   access TP to end hosts.

   A set of "termination-point" indicates all connections of a fabric
   including its internal connections, interconnections with other
   fabrics and also connections to end hosts for a DC network.

   The structure of fabric ports is as follows:

```
augment /nw:networks/nw:network/nw:node/nt:termination-point:
   +--ro fport-attribute
      +--ro name?         string
      +--ro role?         fabric-port-role
      +--ro type?         fabric-port-type
      +--ro device-port?  tp-ref
      +--ro (tunnel-option)?
         +--:(gre)
            +--ro src-ip?       inet:ip-prefix
            +--ro dest-ip?      inet:ip-address
```

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

   New nodes are defined for fabric ports which include name, role of
   the port within the fabric (internal port, external port to outside
   network, access port to end hosts), port type (l2 interface, l3
   interface etc al). By using the device-port defined as a tp-ref,
   this fabric port can be mapped to a device node in the underlay
   network.

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

   The terminiation points information are all learnt from the underlay
   networks but not configured by the fabric topology layer.

4. Fabric YANG Module


```
<CODE BEGINS> file "ietf-fabric-types@2016-09-29.yang"
module ietf-fabric-types {

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

   import ietf-inet-types { prefix "inet"; revision-date "2013-07-15"; }
```

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

     organization
  "IETF I2RS (Interface to the Routing System) Working Group";

     contact
  "WG Web:    <http://tools.ietf.org/wg/i2rs/ >
   WG List:   <mailto:i2rs@ietf.org>

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

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

   Editor:    Yan Zhuang
              <mailto:zhuangyan.zhuang@huawei.com>

   Editor:    Danian Shi
              <mailto:shidanian@huawei.com>";

  description
      "This module contains a collection of YANG definitions for Fabric.";

  revision "2016-09-29" {
      description
          "Initial revision of faas.";
              reference
                     "draft-zhuang-i2rs-yang-dc-fabric-network-topology-02";
  }

  identity fabric-type {
              description
                     "base type for fabric networks";
  }

  identity vxlan-fabric {
      base fabric-type;
              description
                     "vxlan fabric";
  }

  identity vlan-fabric {
      base fabric-type;
              description
                     "vlan fabric";
  }
```

```
    typedef service-capabilities {
        type enumeration {
            enum ip-mapping {
                            description "NAT";
                        }
            enum acl-redirect{
                            description "acl redirect, which can provide SFC
 function";
                        }
            enum dynamic-route-exchange{
                            description "dynamic route exchange";
                        }
        }
                description
                    "capability of the device";
    }

    /*
     * Typedefs
     */
    typedef node-ref {
        type instance-identifier;
        description "A reference to a node in topology";
    }

    typedef tp-ref {
        type instance-identifier;
        description "A reference to a termination point in topology";
    }

    typedef link-ref {
                type instance-identifier;
        description "A reference to a link in topology";
    }

    typedef device-role {
        type enumeration {
            enum SPINE {
                description "a spine node";
            }
            enum LEAF {
                description "a leaf node";
            }
            enum BORDER {
                description "a border node";
            }
        }
        default "LEAF";
        description "device role type";
```

```
        }

    typedef fabric-port-role {
        type enumeration {
            enum internal {
                description "the port used for devices to access each other.";
            }
            enum external {
                description "the port used for fabric to access outside network.
";
            }
            enum access {
                description "the port used for Endpoint to access fabric.";
            }
            enum reserved {
                description " not decided yet. ";
            }
        }
                description "the role of the physical port ";
    }

    typedef fabric-port-type {
        type enumeration {
            enum layer2interface {
                            description "l2 if";
            }
            enum layer3interface {
                            description "l3 if";
            }
            enum layer2Tunnel {
                            description "l2 tunnel";
            }
            enum layer3Tunnel {
                            description "l3 tunnel";
            }
        }
                description
                    "fabric port type";
    }

    typedef underlayer-network-type {
        type enumeration {
            enum VXLAN {
                description "vxlan";
            }
            enum TRILL {
                description "trill";
            }
            enum VLAN {
```

```
                description "vlan";
            }
        }
                description "";
    }

    typedef layer2-protocol-type-enum {
        type enumeration {
            enum VLAN{
                                description "vlan";
                        }
            enum VXLAN{
                                description "vxlan";
                        }
            enum TRILL{
                                description "trill";
                        }
            enum NvGRE{
                                description "nvgre";
                        }
        }
                description "";
    }

    typedef access-type {
        type enumeration {
            enum exclusive{
                                description "exclusive";
                        }
            enum vlan{
                                description "vlan";
                        }
        }
        description "";
    }

    grouping fabric-port {
                description
                        "attributes of a fabric port";
        leaf name {
            type string;
                        description "name of the port";
        }
        leaf role {
            type fabric-port-role;
                        description "role of the port in a fabric";
        }
        leaf type {
```

```
            type fabric-port-type;
                    description "type of the port";
        }
        leaf device-port {
            type tp-ref;
                    description "the device port it mapped to";
        }
        choice tunnel-option {
                    description "tunnel options";

            case gre {
                 leaf src-ip {
                    type inet:ip-prefix;
                                        description "source address";
                 }
                 leaf dest-ip {
                    type inet:ip-address;
                                        description "destination address";
                 }
            }
        }
    }

    grouping route-group {
                description
                    "route attributes";
        list route {
            key "destination-prefix";
                    description "route list";

            leaf description {
                type string;
                description "Textual description of the route.";
            }
            leaf destination-prefix {
                type inet:ipv4-prefix;
                mandatory true;
                description "IPv4 destination prefix.";
            }
            choice next-hop-options {
                            description "choice of next hop options";
                case simple-next-hop {
                    leaf next-hop {
                        type inet:ipv4-address;
                        description "IPv4 address of the next hop.";
                    }
                    leaf outgoing-interface {
                        type nt:tp-id;
```

```
                            description "Name of the outgoing interface.";
                    }
                }
            }
        }
    }

    grouping port-functions {
            description
                    "port functions";

        container port-function {
                    description "port functions";
            choice function-type {
                            description "type of functions";
                case ip-mapping {
                    list ip-mapping-entry {
                                            key "external-ip";
                                            description "list of NAT entry";
                        leaf external-ip {
                            type inet:ipv4-address;
                                            description "external ad
dress";
                        }
                        leaf internal-ip {
                            type inet:ipv4-address;
                                            description "internal ad
dress";
                        }
                    }
                }
            }
        }
    }
    grouping acl-list {
            description "acl list";
        list fabric-acl {
                    key fabric-acl-name;
                    description "fabric acl list";
            leaf fabric-acl-name {
                type string;
                            description "acl name";
            }
        }
    }
}
<CODE ENDS>

<CODE BEGINS> file "ietf-fabric-topology@2017-03-10.yang"
module ietf-fabric-topology {
```

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

       import ietf-network { prefix nw;   }
       import ietf-network-topology { prefix nt;   }
   import ietf-fabric-types { prefix fabrictype; revision-date "2016-09-29"; }

       organization
   "IETF I2RS (Interface to the Routing System) Working Group";

       contact
   "WG Web:    <http://tools.ietf.org/wg/i2rs/ >
    WG List:   <mailto:i2rs@ietf.org>

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

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

    Editor:    Yan Zhuang
               <mailto:zhuangyan.zhuang@huawei.com>

    Editor:    Danian Shi
               <mailto:shidanian@huawei.com>";


   description
       "This module contains a collection of YANG definitions for Fabric.

               Copyright (c) 2016 IETF Trust and the persons identified as
               authors of the code.  All rights reserved.

               Redistribution and use in source and binary forms, with or
               without modification, is permitted pursuant to, and subject
               to the license terms contained in, the Simplified BSD License
               set forth in Section 4.c of the IETF Trust's Legal Provisions
               Relating to IETF Documents
               (http://trustee.ietf.org/license-info).

               This version of this YANG module is part of
               draft-zhuang-i2rs-yang-dc-fabric-network-topology;
               see the RFC itself for full legal notices.";

   revision "2017-03-10" {
               description
                   "remove the rpcs and add extra attributes";
```

```
                    reference
                        "draft-zhuang-i2rs-yang-dc-fabric-network-topology-03";
        }
    revision "2016-09-29" {
        description
            "Initial revision of fabric topology.";
                reference
                    "draft-zhuang-i2rs-yang-dc-fabric-network-topology-02";
    }

    identity fabric-context {
                description
                        "identity of fabric context";
    }

    typedef fabric-id {
        type nw:node-id;
        description
            "An identifier for a fabric in a topology.
            The identifier is generated by compose-fabric RPC.";
    }

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

    grouping fabric-options {
                description "options for a fabric";

        leaf gateway-mode {
            type enumeration {
                enum centralized {
                    description "centerilized gateway";
                }
                enum distributed {
                    description "distributed gateway";
                }
            }
            default "distributed";
                        description "gateway mode";
        }
```

```
        leaf traffic-behavior {
            type enumeration {
                enum normal {
                    description "normal";
                }
                enum policy-driven {
                    description "policy driven";
                }
            }
            default "normal";
                        description "traffic behavior of the fabric";
        }

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

    grouping device-attributes {
                description "device attributes";
        leaf device-ref {
            type fabrictype:node-ref;
                        description
                                "the device it includes to";
        }
        leaf role {
            type fabrictype:device-role;
            default "LEAF";
                        description
                                "role of the node";
        }
    }

    grouping link-attributes {
                description "link attributes";
        leaf link-ref {
            type fabrictype:link-ref;
                        description
                                "the link it includes";
        }
    }

        grouping port-attributes {
                description "port attributes";
        leaf port-ref {
            type fabrictype:tp-ref;
```

```
                             description
                                "port reference";
        }
        leaf port-type {
            type enumeration {
                enum ETH {
                                        description "ETH";
                                }
                enum SERIAL {
                                        description "Serial";
                                }
            }
                        description
                                "port type: ethernet or serial";
        }
        leaf bandwith {
            type enumeration {
                enum 1G {
                                        description "1G";
                                }
                enum 10G {
                                        description "10G";
                                }
                enum 40G {
                                        description "40G";
                                }
                enum 100G {
                                        description "100G";
                                }
                enum 10M {
                                        description "10M";
                                }
                enum 100M {
                                        description "100M";
                                }
                enum 1M {
                                        description "1M";
                                }
            }
                        description
                                "bandwidth on the port";
        }
    }

    grouping fabric-attributes {
                description "attributes of a fabric";

                leaf fabric-id {
```

```
                type fabric-id;
                                description
                                        "fabric id";
        }

        leaf name {
            type string;
                        description
                                "name of the fabric";
        }

        leaf type {
            type fabrictype:underlayer-network-type;
            description
                                "The type of physical network that implements th
is fabric.Examples are vlan, and trill.";
        }

                container vni-capacity {
            description "number of vnis the fabric has";
            leaf min {
                type int32;
                                description
                                        "vni min capacity";
            }

            leaf max {
                type int32;
                                description
                                        "vni max capacity";
            }
        }

        leaf description {
            type string;
                        description
                                "description of the fabric";
        }

        container options {
                        description "options of the fabric";
            uses fabric-options;
        }

        list device-nodes {
                        key device-ref;
                        description "include device nodes in the fabric";
            uses device-attributes;
        }
```

```
      list device-links {
                     key link-ref;
                     description "include device links within the fabric";
         uses link-attributes;
      }

            list device-ports {
         key port-ref;
                     description "include device ports within the fabric";
         uses port-attributes;
      }

   }

      // augment statements

      augment "/nw:networks/nw:network/nw:network-types" {
   description
     "Introduce new network type for Fabric-based logical topology";

            uses fabric-network-type;
      }

   augment "/nw:networks/nw:network/nw:node" {
      when "/nw:networks/nw:network/nw:network-types/fabric-network" {
      description
        "Augmentation parameters apply only for networks
         with fabric topology";
    }
      description "Augmentation for fabric nodes created by faas.";

      container fabric-attribute {
                     description
                     "attributes for a fabric network";

         uses fabric-attributes;
      }
   }

   augment "/nw:networks/nw:network/nw:node/nt:termination-point" {
      when "/nw:networks/nw:network/nw:network-types/fabric-network" {
      description
        "Augmentation parameters apply only for networks
         with fabric topology";
    }
      description "Augmentation for port on fabric.";

      container fport-attribute {
```

```
            config false;
                        description
                        "attributes for fabric ports";
            uses fabrictype:fabric-port;
        }
    }
}
<CODE ENDS>
```

5. Security Consideration

    TBD

6. Acknowledgements

7. References

7.1. Normative References

   [I-D.draft-ietf-i2rs-yang-l3-topology]
            Clemm, A., Medved, J., Tkacik, T., Liu, X., Bryskin, I.,
            Guo, A., Ananthakrishnan, H., Bahadur, N., and V. Beeram,
            "A YANG Data Model for Layer 3 Topologies", I-D draft-
            ietf-i2rs-yang-l3-topology-04, September 2016.

   [I-D.draft-ietf-i2rs-yang-network-topo]
            Clemm, A., Medved, J., Tkacik, T., Varga, R., Bahadur, N.,
            and H. Ananthakrishnan, "A YANG Data Model for Network
            Topologies", I-D draft-ietf-i2rs-yang-network-topo-06,
            September 2016.

   [I-D.draft-ietf-nvo3-vxlan-gpe]
            Maino, F., Kreeger, L., and U. Elzur, "Generic Protocol
            Extension for VXLAN", I-D draft-ietf-i2rs-yang-network-
            topo-02, October 2016.

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

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

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

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

7.2. Informative References

   [I-D.draft-ietf-i2rs-usecase-reqs-summary]
              Hares, S. and M. Chen, "Summary of I2RS Use Case
              Requirements", I-D draft-ietf-i2rs-usecase-reqs-summary-
              01, May 2015.

Appendix A. Non NMDA -state modules

        <CODE BEGINS> file "ietf-fabric-topology-state@2017-06-29.yang"
        module ietf-fabric-topology-state {

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

        import ietf-network { prefix nw; }
    import ietf-fabric-types { prefix fabrictype; revision-date "2016-09-29"; }
        import ietf-fabric-topology {prefix fp;}
        organization
    "IETF I2RS (Interface to the Routing System) Working Group";

        contact
        "WG Web: <http://tools.ietf.org/wg/i2rs/ >
     WG List: <mailto:i2rs@ietf.org>

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

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

      Editor: Yan Zhuang
                <mailto:zhuangyan.zhuang@huawei.com>

      Editor: Danian Shi
                <mailto:shidanian@huawei.com>";

    description
        "This module contains a collection of YANG definitions for Fabric topolo
gy state for non NMDA.

                Copyright (c) 2016 IETF Trust and the persons identified as
                authors of the code. All rights reserved.

```
     revision "2017-06-29"{
                description
                        "update to NMDA compliant format";
                reference
                        "draft-zhuang-i2rs-yang-dc-fabric-network-topology-04";
        }

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

    grouping fabric-options {
                description "options for a fabric";

        leaf gateway-mode {
            type enumeration {
                enum centralized {
                    description "centerilized gateway";
                }
                enum distributed {
                    description "distributed gateway";
                }
            }
            default "distributed";
                    description "gateway mode";
        }

        leaf traffic-behavior {
            type enumeration {
                enum normal {
```

```
                    description "normal";
                }
                enum policy-driven {
                    description "policy driven";
                }
            }
            default "normal";
                    description "traffic behavior of the fabric";
        }

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

    grouping device-attributes {
                description "device attributes";
        leaf device-ref {
            type fabrictype:node-ref;
                    description
                            "the device it includes to";
        }
        leaf role {
            type fabrictype:device-role;
            default "LEAF";
                    description
                            "role of the node";
        }
    }

    grouping link-attributes {
                description "link attributes";
        leaf link-ref {
            type fabrictype:link-ref;
                    description
                            "the link it includes";
        }
    }

        grouping port-attributes {
                description "port attributes";
        leaf port-ref {
            type fabrictype:tp-ref;
                    description
                            "port reference";
        }
```

```
      leaf port-type {
          type enumeration {
              enum ETH {
                                      description "ETH";
                            }
              enum SERIAL {
                                      description "Serial";
                            }
          }
                    description
                            "port type: ethernet or serial";
      }
      leaf bandwith {
          type enumeration {
              enum 1G {
                                      description "1G";
                            }
              enum 10G {
                                      description "10G";
                            }
              enum 40G {
                                      description "40G";
                            }
              enum 100G {
                                      description "100G";
                            }
              enum 10M {
                                      description "10M";
                            }
              enum 100M {
                                      description "100M";
                            }
              enum 1M {
                                      description "1M";
                            }
          }
                    description
                            "bandwidth on the port";
      }
  }

  grouping fabric-attributes {
            description "attributes of a fabric";

            leaf fabric-id {
            type fp:fabric-id;
                          description
                                  "fabric id";
```

```
        }

        leaf name {
            type string;
                        description
                                "name of the fabric";
        }

        leaf type {
            type fabrictype:underlayer-network-type;
            description
                                    "The type of physical network that implements th
is fabric.Examples are vlan, and trill.";
        }

                container vni-capacity {
            description "number of vnis the fabric has";
            leaf min {
                type int32;
                                description
                                        "vni min capacity";
            }

            leaf max {
                type int32;
                                description
                                        "vni max capacity";
            }
        }

        leaf description {
            type string;
                        description
                                "description of the fabric";
        }

        container options {
                        description "options of the fabric";
            uses fabric-options;
        }

        list device-nodes {
                        key device-ref;
                        description "include device nodes in the fabric";
            uses device-attributes;
        }

        list device-links {
                        key link-ref;
```

```
                        description "include device links within the fabric";
            uses link-attributes;
        }

                list device-ports {
            key port-ref;
                        description "include device ports within the fabric";
            uses port-attributes;
        }

    }

        // augment statements

        augment "/nw:networks/nw:network/nw:network-types" {
    description
      "Introduce new network type for Fabric-based logical topology";

                uses fabric-network-type;
        }

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

        container fabric-attribute-state {
                        config false;
                        description
                        "attributes for a fabric network";

            uses fabric-attributes;
        }
    }
}

                <CODE ENDS>
```

Authors' Addresses

Yan Zhuang
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: zhuangyan.zhuang@huawei.com

Danian Shi
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: shidanian@huawei.com

Rong Gu
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing, Beijing 100053
China

Email: gurong_cmcc@outlook.com

Hariharan Ananthakrishnan
Packet Design

Email: hari@packetdesign.com

Network Working Group   A. Clemm
Internet-Draft   Huawei USA
Intended status: Standards Track   J. Medved
Expires: June 19, 2018   Cisco
                                                                R. Varga
                                              Pantheon Technologies SRO
                                                                 X. Liu
                                                                  Jabil
                                                      H. Ananthakrishnan
                                                           Packet Design
                                                              N. Bahadur
                                                       Bracket Computing
                                                       December 16, 2017

                  A YANG Data Model for Layer 3 Topologies
                    draft-ietf-i2rs-yang-l3-topology-16.txt

Abstract

   This document defines a YANG data model for layer 3 network
   topologies.

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 June 19, 2018.

publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction

   This document introduces a YANG [RFC7950] [RFC6991] data model for
   Layer 3 network topologies, specifically Layer 3 Unicast. The model
   allows an application to have a holistic view of the topology of a
   Layer 3 network, all contained in a single conceptual YANG datastore.
   The data model builds on top of, and augments, the data model for
   network topologies defined in
   [I-D.draft-ietf-i2rs-yang-network-topo].

   This document also shows how the model can be further refined to
   cover different Layer 3 Unicast topology types. For this purpose, an
   example model is introduced that covers OSPF [RFC2328]. This example
   is intended purely for illustrative purpose; we expect that a
   complete OSPF model will be more comprehensive and refined than the
   example shown here.

There are multiple applications for a topology data model. A number
of use cases have been defined in section 6 of
[I-D.draft-ietf-i2rs-usecase-reqs-summary]. For example, nodes
within the network can use the data model to capture their
understanding of the overall network topology and expose it to a
network controller. A network controller can then use the
instantiated topology data to compare and reconcile its own view of
the network topology with that of the network elements that it
controls. Alternatively, nodes within the network could propagate
this understanding to compare and reconcile this understanding either
amongst themselves or with help of a controller. Beyond the network
element itself, a network controller might even use the data model to
represent its view of the topology that it controls and expose it to
applications north of itself.

The data model for Layer 3 Unicast topologies defined in this
document is specified in a YANG module "ietf-l3-unicast-topology".
To do so, it augments the general network topology model defined in
[I-D.draft-ietf-i2rs-yang-network-topo] with information specific to
Layer 3 Unicast. This way, the general topology model is extended to
be able to meet the needs of Layer 3 Unicast topologies.

Information that is kept in the Traffic Engineering Database (TED)
will be specified in a separate model
[I-D.draft-ietf-teas-yang-te-topo] and outside the scope of this
specification.

2. Key Words

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.

3. Definitions and Acronyms

As this document defines a YANG data model, in this document many
terms are used that have been defined in conjunction with YANG
[RFC7950] and NETCONF [RFC6241]. Some terms, such as datastore and
data tree, are repeated here for clarity and to put them in context.

Datastore: A conceptual place to store and access information. A
datastore might be implemented, for example, using files, a database,
flash memory locations, or combinations thereof. A datastore maps to
an instantiated YANG data tree. (Definition adopted from
[I-D.draft-ietf-netmod-revised-datastores])

Data subtree: An instantiated data node and the data nodes that are
hierarchically contained within it.

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

LSP: Label Switched Path

NETCONF: Network Configuration Protocol

NMDA: Network Management Datastore Architecture

OSPF: Open Shortest Path First, a link state routing protocol

URI: Uniform Resource Identifier

SRLG: Shared Risk Link Group

TED: Traffic Engineering Database

YANG: YANG is a data modeling language used to model configuration
data, state data, Remote Procedure Calls, and notifications for
network management protocols [RFC7950]

4. Model Structure

The Layer 3 Unicast topology model is defined by YANG module "l3-
unicast-topology". The relationship of this module with other YANG
modules is roughly depicted in the figure below.

```
            +-----------------------------+
            |  +----------------------+   |
            |  |     ietf-network     |   |
            |  +----------^-----------+   |
            |             |               |
            |  +----------------------+   |
            |  | ietf-network-topology |  |
            |  +----------+-----------+   |
            +------------^---------------+
                         |
                         |
            +------------^-------------+
            | ietf-l3-unicast-topology |
            +------------^-------------+
                         |
                         |
              +----------^-----------+
              | example-ospf-topology |
              +----------------------+
```
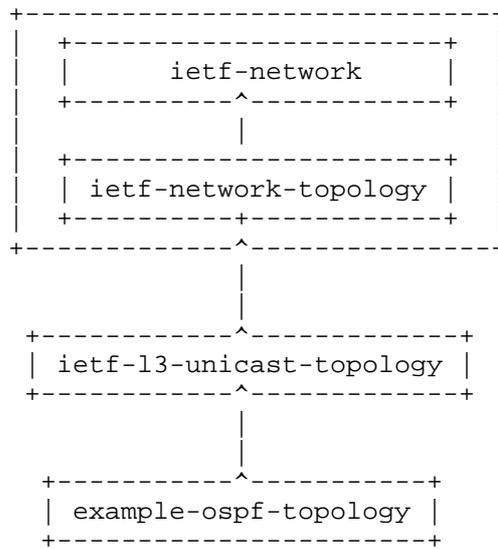
Figure 1: Overall model structure

YANG modules "ietf-network" and "ietf-network-topology" collectively
define the basic network topology model
[I-D.draft-ietf-i2rs-yang-network-topo]. YANG module "ietf-l3-
unicast-topology" augments those models with additional definitions
needed to represent Layer 3 Unicast topologies. This module in turn
can be augmented by YANG modules with additional definitions for
specific types of Layer 3 Unicast topologies, such as OSPF and for
IS-IS topologies.

The YANG modules ietf-network and ietf-network-topology are designed
to be used in conjunction with implementations that support the
Network Management Datastore Architecture (NMDA) defined in
[I-D.draft-ietf-netmod-revised-datastores]. Accordingly, the same is
true for the YANG modules that augment it. In order to allow
implementations to use the model even in cases when NMDA is not
supported, companion YANG modules (that SHOULD NOT be supported by
implementations that support NMDA) are defined in an Appendix, see
Appendix A.

5. Layer 3 Unicast Topology Model Overview

   The Layer 3 Unicast topology model is defined by YANG module "ietf-
   l3-unicast-topology". Its structure is depicted in the following
   diagram. The notation syntax follows
   [I-D.draft-ietf-netmod-yang-tree-diagrams]. For purposes of brevity,
   notifications are not depicted.

```
module: ietf-l3-unicast-topology
  augment /nw:networks/nw:network/nw:network-types:
    +--rw l3-unicast-topology!
  augment /nw:networks/nw:network:
    +--rw l3-topology-attributes
       +--rw name?    string
       +--rw flag*    l3-flag-type
  augment /nw:networks/nw:network/nw:node:
    +--rw l3-node-attributes
       +--rw name?         inet:domain-name
       +--rw flag*         node-flag-type
       +--rw router-id*    rt-types:router-id
       +--rw prefix* [prefix]
          +--rw prefix    inet:ip-prefix
          +--rw metric?   uint32
          +--rw flag*     prefix-flag-type
  augment /nw:networks/nw:network/nt:link:
    +--rw l3-link-attributes
       +--rw name?     string
       +--rw flag*     link-flag-type
       +--rw metric1?  uint64
       +--rw metric2?  uint64
  augment /nw:networks/nw:network/nw:node/nt:termination-point:
    +--rw l3-termination-point-attributes
       +--rw (termination-point-type)?
          +--:(ip)
          |  +--rw ip-address*       inet:ip-address
          +--:(unnumbered)
          |  +--rw unnumbered-id?    uint32
          +--:(interface-name)
             +--rw interface-name?   string
```

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

o A new network topology type is introduced, l3-unicast-topology. The corresponding container augments the network-types of the ietf-network module.

o Additional topology attributes are introduced, defined in a grouping, which augments the "network" list of the network module. The attributes include a name for the topology, as well as a set of flags (represented through a leaf-list). Each type of flag is represented by a separate identity. This allows to introduce additional flags in augmenting modules using additional identities without needing to revise this module.

    o Additional data objects for nodes are introduced by augmenting the
       "node" list of the network module. New objects include again a
       set of flags, as well as a list of prefixes. Each prefix in turn
       includes an ip prefix, a metric, and a prefix-specific set of
       flags.

    o Links (in the ietf-network-topology module) are augmented with a
       set of parameters as well, allowing to associate a link with a
       link name, another set of flags, and a link metric.

    o Termination points (in the ietf-network-topology module as well)
       are augmented with a choice of IP address, identifier, or name.

In addition, the module defines a set of notifications to alert
clients of any events concerning links, nodes, prefixes, and
termination points. Each notification includes an indication of the
type of event, the topology from which it originated, and the
affected node, or link, or prefix, or termination point. In
addition, as a convenience to applications, additional data of the
affected node, or link, or termination point (respectively) is
included. While this makes notifications larger in volume than they
would need to be, it avoids the need for subsequent retrieval of
context information, which also might have changed in the meantime.

6. Layer 3 Unicast Topology YANG Module

```
<CODE BEGINS> file "ietf-l3-unicast-topology@2017-12-16.yang"
module ietf-l3-unicast-topology {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-l3-unicast-topology";
  prefix "l3t";
  import ietf-network {
    prefix "nw";
  }
  import ietf-network-topology {
    prefix "nt";
  }
  import ietf-inet-types {
    prefix "inet";
  }
  import ietf-routing-types {
    prefix "rt-types";
  }
  organization
    "IETF I2RS (Interface to the Routing System) Working Group";
  contact
    "WG Web:    <http://tools.ietf.org/wg/i2rs/>
```

```
     WG List:    <mailto:i2rs@ietf.org>
     Editor:     Alexander Clemm
                 <mailto:ludwig@clemm.org>
     Editor:     Jan Medved
                 <mailto:jmedved@cisco.com>
     Editor:     Robert Varga
                 <mailto:robert.varga@pantheon.tech>
     Editor:     Xufeng Liu
                 <mailto:xliu@kuatrotech.com>
     Editor:     Nitin Bahadur
                 <mailto:nitin_bahadur@yahoo.com>
     Editor:     Hariharan Ananthakrishnan
                 <mailto:hari@packetdesign.com>";
   description
     "This module defines a model for Layer 3 Unicast
     topologies.
     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
     draft-ietf-i2rs-yang-l3-topology-16;
     see the RFC itself for full legal notices.
     NOTE TO RFC EDITOR: Please replace above reference to
     draft-ietf-i2rs-yang-l3-topology-16 with RFC
     number when published (i.e. RFC xxxx).";
   revision "2017-12-16" {
     description
       "Initial revision.
        NOTE TO RFC EDITOR: Please replace the following reference
        to draft-ietf-i2rs-yang-l3-topology-16 with
        RFC number when published (i.e. RFC xxxx).";
     reference
       "draft-ietf-i2rs-yang-l3-topology-16";
   }

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

   typedef l3-event-type {
     type enumeration {
       enum "add" {
         description
```

```
              "An Layer 3 node or link or prefix or termination-point has
              been added";
          }
          enum "remove" {
            description
              "An Layer 3 node or link or prefix or termination-point has
              been removed";
          }
          enum "update" {
            description
              "An Layer 3 node or link or prefix or termination-point has
              been updated";
          }
        }
        description "Layer 3 Event type for notifications";
      }

      typedef prefix-flag-type {
        type identityref {
          base "flag-identity";
        }
        description "Prefix flag attributes";
      }

      typedef node-flag-type {
        type identityref {
          base "flag-identity";
        }
        description "Node flag attributes";
      }

      typedef link-flag-type {
        type identityref {
          base "flag-identity";
        }
        description "Link flag attributes";
      }

      typedef l3-flag-type {
        type identityref {
          base "flag-identity";
        }
        description "L3 flag attributes";
      }

      grouping l3-prefix-attributes {
        description
          "L3 prefix attributes";
```

```
      leaf prefix {
        type inet:ip-prefix;
        description
          "IP prefix value";
      }
      leaf metric {
        type uint32;
        description
          "Prefix metric";
      }
      leaf-list flag {
        type prefix-flag-type;
        description
          "Prefix flags";
      }
    }
    grouping l3-unicast-topology-type {
      description "Identify the topology type to be L3 unicast.";
      container l3-unicast-topology {
        presence "indicates L3 Unicast Topology";
        description
          "The presence of the container node indicates L3 Unicast
          Topology";
      }
    }
    grouping l3-topology-attributes {
      description "Topology scope attributes";
      container l3-topology-attributes {
        description "Containing topology attributes";
        leaf name {
          type string;
          description
            "Name of the topology";
        }
        leaf-list flag {
          type l3-flag-type;
          description
            "Topology flags";
        }
      }
    }
    grouping l3-node-attributes {
      description "L3 node scope attributes";
      container l3-node-attributes {
        description
          "Containing node attributes";
        leaf name {
          type inet:domain-name;
```

```
              description
                "Node name";
            }
            leaf-list flag {
              type node-flag-type;
              description
                "Node flags";
            }
            leaf-list router-id {
              type rt-types:router-id;
              description
                "Router-id for the node";
            }
            list prefix {
              key "prefix";
              description
                "A list of prefixes along with their attributes";
              uses l3-prefix-attributes;
            }
          }
        }
      }
      grouping l3-link-attributes {
        description
          "L3 link scope attributes";
        container l3-link-attributes {
          description
            "Containing link attributes";
          leaf name {
            type string;
            description
              "Link Name";
          }
          leaf-list flag {
            type link-flag-type;
            description
              "Link flags";
          }
          leaf metric1 {
            type uint64;
            description
                "Link Metric 1";
          }
          leaf metric2 {
            type uint64;
            description
                "Link Metric 2";
          }
        }
```

```
        }
      grouping l3-termination-point-attributes {
         description "L3 termination point scope attributes";
         container l3-termination-point-attributes {
           description
             "Containing termination point attributes";
           choice termination-point-type {
             description
               "Indicates the termination point type";
             case ip {
               leaf-list ip-address {
                 type inet:ip-address;
                 description
                   "IPv4 or IPv6 address.";
               }
             }
             case unnumbered {
               leaf unnumbered-id {
                 type uint32;
                 description
                   "Unnumbered interface identifier.
                   The identifier will correspond to the ifIndex value
                   of the interface, i.e. the ifIndex value of the
                   ifEntry that represents the interface in
                   implementations where the Interfaces Group MIB
                   (RFC 2863) is supported.";
                 reference
                   "RFC 2863: The Interfaces Group MIB";
               }
             }
             case interface-name {
               leaf interface-name {
                 type string;
                 description
                   "A name of the interface.  The name can (but does not
                   have to) correspond to an interface reference of a
                   containing node's interface, i.e. the path name of a
                   corresponding interface data node on the containing
                   node reminiscent of data type if-ref defined in
                   RFC 7223. It should be noted that data type if-ref of
                   RFC 7223 cannot be used directly, as this data type
                   is used to reference an interface in a datastore of
                   a single node in the network, not to uniquely
                   reference interfaces across a network.";
               }
             }
           }
         }
```

```
      }
      augment "/nw:networks/nw:network/nw:network-types" {
        description
          "Introduce new network type for L3 unicast topology";
        uses l3-unicast-topology-type;
      }
      augment "/nw:networks/nw:network" {
        when "nw:network-types/l3t:l3-unicast-topology" {
          description
            "Augmentation parameters apply only for networks with
            L3 unicast topology";
        }
        description
            "L3 unicast for the network as a whole";
        uses l3-topology-attributes;
      }
      augment "/nw:networks/nw:network/nw:node" {
        when "../nw:network-types/l3t:l3-unicast-topology" {
          description
            "Augmentation parameters apply only for networks with
            L3 unicast topology";
        }
        description
            "L3 unicast node level attributes ";
        uses l3-node-attributes;
      }
      augment "/nw:networks/nw:network/nt:link" {
        when "../nw:network-types/l3t:l3-unicast-topology" {
          description
            "Augmentation parameters apply only for networks with
            L3 unicast topology";
        }
        description
          "Augment topology link attributes";
        uses l3-link-attributes;
      }
      augment "/nw:networks/nw:network/nw:node/"
            +"nt:termination-point" {
        when "../../nw:network-types/l3t:l3-unicast-topology" {
          description
            "Augmentation parameters apply only for networks with
            L3 unicast topology";
        }
        description "Augment topology termination point configuration";
        uses l3-termination-point-attributes;
      }
      notification l3-node-event {
        description
```

```
          "Notification event for L3 node";
        leaf l3-event-type {
          type l3-event-type;
          description
            "Event type";
        }
        uses nw:node-ref;
        uses l3-unicast-topology-type;
        uses l3-node-attributes;
      }
      notification l3-link-event {
        description
          "Notification event for L3 link";
        leaf l3-event-type {
          type l3-event-type;
          description
            "Event type";
        }
        uses nt:link-ref;
        uses l3-unicast-topology-type;
        uses l3-link-attributes;
      }
      notification l3-prefix-event {
        description
          "Notification event for L3 prefix";
        leaf l3-event-type {
          type l3-event-type;
          description
            "Event type";
        }
        uses nw:node-ref;
        uses l3-unicast-topology-type;
        container prefix {
          description
            "Containing L3 prefix attributes";
          uses l3-prefix-attributes;
        }
      }
      notification termination-point-event {
        description
          "Notification event for L3 termination point";
        leaf l3-event-type {
          type l3-event-type;
          description
            "Event type";
        }
        uses nt:tp-ref;
        uses l3-unicast-topology-type;
```

```
      uses l3-termination-point-attributes;
    }
  }

  <CODE ENDS>
```

7. Interactions with Other YANG Modules

   As described in section Section 4, the model builds on top of, and
   augments, the YANG modules defined in
   [I-D.draft-ietf-i2rs-yang-network-topo]. Specifically, module ietf-
   l3-unicast-topology augments modules "ietf-network" and "ietf-
   network-topology". In addition, the model makes use of data types
   that have been defined in [RFC6991].

   The model defines a protocol independent YANG data model with layer 3
   topology information. It is separate from and not linked with data
   models that are used to configure routing protocols or routing
   information. This includes e.g. model "ietf-routing" [RFC8022] and
   model "ietf-fb-rib" [I-D.draft-acee-rtgwg-yang-rib-extend]. That
   said, the model does import a type definition from model "ietf-
   routing-types" [RFC8294].

   The model obeys the requirements for the ephemeral state found in the
   document [RFC8242]. For ephemeral topology data that is server
   provided, the process tasked with maintaining topology information
   will load information from the routing process (such as OSPF) into
   the data model without relying on a configuration datastore.

8. IANA Considerations

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

   URI: urn:ietf:params:xml:ns:yang:ietf-l3-unicast-topology
   Registrant Contact: The IESG.
   XML: N/A; the requested URI is an XML namespace.

   URI: urn:ietf:params:xml:ns:yang:ietf-l3-unicast-topology-state
   Registrant Contact: The IESG.
   XML: N/A; the requested URI is an XML namespace.

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

   Name: ietf-l3-unicast-topology
   Namespace: urn:ietf:params:xml:ns:yang:ietf-l3-unicast-topology
   Prefix: l3t

Reference: draft-ietf-i2rs-yang-l3-topology-16.txt (RFC form)

Name: ietf-l3-unicast-topology-state
Namespace: urn:ietf:params:xml:ns:yang:ietf-l3-unicast-topology-state
Prefix: l3t-s
Reference: draft-ietf-i2rs-yang-l3-topology-16.txt (RFC form)

9. Security Considerations

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

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

In general, Layer 3 Unicast topologies are system-controlled and
provide ephemeral topology information. In an NMDA-complient server,
they are only part of <operational> which provides read-only access
to clients, they are less vulnerable. That said, the YANG module
does in principle allow information to be configurable.

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

   l3-topology-attributes: A malicious client could attempt to
   sabotage the configuration of any of the ctonained attributes,
   i.e. the name or the flag data nodes.

   l3-node-attributes: A malicious client could attempt to sabotage
   the configuration of important node attributes, such as the
   router-id or node prefix.

   l3-link-attributes: A malicious client could attempt to sabotage
   the configuration of important link attributes, such as name,
   flag, and metrics of the link respectively corresponding data
   nodes.

l3-termination-point-attributes: A malicious client could attempt to sabotage the configuration information of a termination point, such as its ip-address and interface name, respectively the corresponding data nodes.

10. Contributors

The model presented in this document was contributed to by more people than can be listed on the author list. Additional contributors include:

o Vishnu Pavan Beeram, Juniper

o Igor Bryskin, Huawei

o Ken Gray, Cisco

o Aihua Guo, Huawei

o Tom Nadeau, Brocade

o Tony Tkacik

o Aleksandr Zhdankin, Cisco

11. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Alia Atlas, Andy Bierman, Benoit Claise, Joel Halpern, Susan Hares, Ladislav Lhotka, Carl Moberg, Carlos Pignataro, Juergen Schoenwaelder, Michal Vasco, and Kent Watsen.

12. References

12.1. Normative References

[I-D.draft-ietf-i2rs-yang-network-topo]
          Clemm, A., Medved, J., Varga, R., Bahadur, N.,
          Ananthakrishnan, H., and X. Liu, "A YANG Data Model for
          Network Topologies", I-D draft-ietf-i2rs-yang-network-
          topo-19, December 2017.

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

   [RFC1195] Callon, R., "Use of OSI IS-IS for Routing in TCP/IP and
              Dual Environments", RFC 1195, December 1990.

   [RFC2119] Bradner, S., "Key words for use in RFCs to indicate
              requirement levels", RFC 2119, March 1997.

   [RFC2328] Moy, J., "OSPF Version 2", RFC 2328, April 1998.

   [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group
              MIB", RFC 2863, June 2000.

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

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246, August 2008.

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

   [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A.
              Bierman, "Network Configuration Protocol (NETCONF)",
              RFC 6241, June 2011.

   [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure
              Shell (SSH)", RFC 6242, June 2011.

   [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration
              Protocol (NETCONF) Access Control Model", RFC 6536, March
              2012.

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

   [RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language",
              RFC 7950, August 2016.

   [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG",
              RFC 7951, August 2016.

   [RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger,
              "Common YANG Data Types for the Routing Area", RFC 8294,
              December 2014.

12.2. Informative References

   [I-D.draft-acee-rtgwg-yang-rib-extend]
             Lindem, A. and Y. Qu, "YANG Data Model for RIB
             Extensions", I-D draft-acee-rtgwg-yang-rib-extend-05,
             October 2017.

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

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

   [I-D.draft-ietf-teas-yang-te-topo]
             Liu, X., Bryskin, I., Beeram, V., Saad, T., Shah, H., and
             O. Gonzalez De Dios, "YANG Data Model for TE Topologies",
             I-D draft-ietf-teas-yang-te-topo-13, October 2017.

   [RFC7223] Bjorklund, M., "A YANG Data Model for Routing Management",
             RFC 7223, May 2014.

   [RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing
             Management", RFC 8022, November 2016.

   [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF
             Protocol", RFC 8040, January 2017.

   [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
             2119 Key Words", RFC 8174, May 2017.

   [RFC8242] Haas, J. and S. Hares, "I2RS Ephemeral State
             Requirements", RFC 8242, September 2017.

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

   The YANG module ietf-l3-unicast-topology defined in this document
   augments two modules, ietf-network and ietf-network-topology, that
   are designed to be used in conjunction with implementations that
   support the Network Management Datastore Architecture (NMDA) defined
   in [I-D.draft-ietf-netmod-revised-datastores]. In order to allow
   implementations to use the model even in cases when NMDA is not
   supported, a set of companion modules have been defined that
   represent a state model of networks and network topologies, ietf-
   network-state and ietf-network-topology-state, respectively.

   In order to be able to use the model for layer 3 topologies defined
   in this in this document in conjunction with non-NMDA compliant
   implementations, a corresponding companion module needs to be
   introduced as well. This companion module, ietf-l3-unicast-topology-
   state, mirrors ietf-l3-unicast-topology. However, the module
   augments ietf-network-state and ietf-network-topology-state (instead
   of ietf-network and ietf-network-topology) and all of its data nodes
   are non-configurable.

   Similar considerations apply for any modules that augment ietf-l3-
   unicast-topology, such as the example modules defined in see
   Appendix B, example-ospf-topology. For non-NMDA compliant
   implementations, companion modules will need to be introduced that
   represent state information and are non-configurable, augmenting
   ietf-l3-unicast-topology-state instead of ietf-l3-unicast-topology.
   Because they served as examples only, companion modules for those
   examples are not given.

   Like ietf-network-state and ietf-network-topology-state, ietf-l3-
   unicast-topology SHOULD NOT be supported by implementations that
   support NMDA. It is for this reason that the module is defined in
   the Appendix.

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

   <CODE BEGINS> file "ietf-l3-unicast-topology-state@2017-12-16.yang"
   module ietf-l3-unicast-topology-state {
     yang-version 1.1;
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-l3-unicast-topology-state";
     prefix "l3t-s";
     import ietf-network-state {
       prefix "nw-s";
     }

```
import ietf-network-topology-state {
  prefix "nt-s";
}
import ietf-l3-unicast-topology {
  prefix "l3t";
}
organization
  "IETF I2RS (Interface to the Routing System) Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/i2rs/>
   WG List:   <mailto:i2rs@ietf.org>
   Editor:    Alexander Clemm
              <mailto:ludwig@clemm.org>
   Editor:    Jan Medved
              <mailto:jmedved@cisco.com>
   Editor:    Robert Varga
              <mailto:robert.varga@pantheon.tech>
   Editor:    Xufeng Liu
              <mailto:xliu@kuatrotech.com>
   Editor:    Nitin Bahadur
              <mailto:nitin_bahadur@yahoo.com>
   Editor:    Hariharan Ananthakrishnan
              <mailto:hari@packetdesign.com>";
description
  "This module defines a model for Layer 3 Unicast topology
   state, representing topology that is either learned, or topology
   that results from applying topology that has been configured per
   the ietf-l3-unicast-topology model, mirroring the corresponding
   data nodes in this model.

   The model mirrors ietf-l3-unicast-topology, but contains only
   read-only state data.  The model is not needed when the
   underlying implementation infrastructure supports the Network
   Management Datastore Architecture (NMDA).

   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
   draft-ietf-i2rs-yang-l3-topology-16;
   see the RFC itself for full legal notices.
   NOTE TO RFC EDITOR: Please replace above reference to
   draft-ietf-i2rs-yang-l3-topology-16 with RFC
```

```
      number when published (i.e. RFC xxxx).";
    revision "2017-12-16" {
      description
        "Initial revision.
         NOTE TO RFC EDITOR: Please replace the following reference
         to draft-ietf-i2rs-yang-l3-topology-16 with
         RFC number when published (i.e. RFC xxxx).";
      reference
        "draft-ietf-i2rs-yang-l3-topology-16";
    }
    augment "/nw-s:networks/nw-s:network/nw-s:network-types" {
      description
        "Introduce new network type for L3 unicast topology";
      uses l3t:l3-unicast-topology-type;
    }
    augment "/nw-s:networks/nw-s:network" {
      when "nw-s:network-types/l3t-s:l3-unicast-topology" {
        description
          "Augmentation parameters apply only for networks with
          L3 unicast topology";
      }
      description
          "L3 unicast for the network as a whole";
      uses l3t:l3-topology-attributes;
    }
    augment "/nw-s:networks/nw-s:network/nw-s:node" {
      when "../nw-s:network-types/l3t-s:l3-unicast-topology" {
        description
          "Augmentation parameters apply only for networks with
          L3 unicast topology";
      }
      description
          "L3 unicast node level attributes ";
      uses l3t:l3-node-attributes;
    }
    augment "/nw-s:networks/nw-s:network/nt-s:link" {
      when "../nw-s:network-types/l3t-s:l3-unicast-topology" {
        description
          "Augmentation parameters apply only for networks with
          L3 unicast topology";
      }
      description
        "Augment topology link attributes";
      uses l3t:l3-link-attributes;
    }
    augment "/nw-s:networks/nw-s:network/nw-s:node/"
          +"nt-s:termination-point" {
      when "../../nw-s:network-types/l3t-s:l3-unicast-topology" {
```

```
        description
          "Augmentation parameters apply only for networks with
          L3 unicast topology";
      }
      description "Augment topology termination point configuration";
      uses l3t:l3-termination-point-attributes;
    }
    notification l3-node-event {
      description
        "Notification event for L3 node";
      leaf l3-event-type {
        type l3t:l3-event-type;
        description
          "Event type";
      }
      uses nw-s:node-ref;
      uses l3t:l3-unicast-topology-type;
      uses l3t:l3-node-attributes;
    }
    notification l3-link-event {
      description
        "Notification event for L3 link";
      leaf l3-event-type {
        type l3t:l3-event-type;
        description
          "Event type";
      }
      uses nt-s:link-ref;
      uses l3t:l3-unicast-topology-type;
      uses l3t:l3-link-attributes;
    }
    notification l3-prefix-event {
      description
        "Notification event for L3 prefix";
      leaf l3-event-type {
        type l3t:l3-event-type;
        description
          "Event type";
      }
      uses nw-s:node-ref;
      uses l3t:l3-unicast-topology-type;
      container prefix {
        description
          "Containing L3 prefix attributes";
        uses l3t:l3-prefix-attributes;
      }
    }
    notification termination-point-event {
```

```
      description
        "Notification event for L3 termination point";
      leaf l3-event-type {
        type l3t:l3-event-type;
        description
          "Event type";
      }
      uses nt-s:tp-ref;
      uses l3t:l3-unicast-topology-type;
      uses l3t:l3-termination-point-attributes;
    }
  }
```

    <CODE ENDS>

Appendix B. Extending the Model

   The model can be extended for specific Layer 3 Unicast types.
   Examples include OSPF and IS-IS topologies. In the following, one
   additional YANG module is introduced that define simple topology
   model for OSPF. This module is intended to serve as an example that
   illustrates how the general topology model can be refined across
   multiple levels. It does not constitute full-fledged OSPF topology
   model which may be more comprehensive and refined than the model that
   is described here.

B.1. Example OSPF Topology

B.1.1. Model Overview

   The following model shows how the Layer 3 Unicast topology model can
   be extended, in this case to cover OSFP topologies. For this
   purpose, a set of augmentations are introduced in a separate YANG
   module, "example-ospf-topology", whose structure is depicted in the
   following diagram. As before, the notation syntax follows
   [I-D.draft-ietf-netmod-yang-tree-diagrams].

```
module: example-ospf-topology
augment /nw:networks/nw:network/nw:network-types/l3t:l3-unicast-topology:
  +--rw ospf!
augment /nw:networks/nw:network/l3t:l3-topology-attributes:
  +--rw ospf-topology-attributes
     +--rw area-id?   area-id-type
augment /nw:networks/nw:network/nw:node/l3t:l3-node-attributes:
  +--rw ospf-node-attributes
     +--rw (router-type)?
     |  +--:(abr)
     |  |  +--rw abr?             empty
     |  +--:(asbr)
     |  |  +--rw asbr?            empty
     |  +--:(internal)
     |  |  +--rw internal?        empty
     |  +--:(pseudonode)
     |     +--rw pseudonode?      empty
     +--rw dr-interface-id?   uint32
augment /nw:networks/nw:network/nt:link/l3t:l3-link-attributes:
  +--rw ospf-link-attributes
augment /l3t:l3-node-event:
  +---- ospf!
  +---- ospf-node-attributes
     +---- (router-type)?
     |  +--:(abr)
     |  |  +---- abr?             empty
     |  +--:(asbr)
     |  |  +---- asbr?            empty
     |  +--:(internal)
     |  |  +---- internal?        empty
     |  +--:(pseudonode)
     |     +---- pseudonode?      empty
     +---- dr-interface-id?   uint32
augment /l3t:l3-link-event:
  +---- ospf!
  +---- ospf-link-attributes
```

The module augments "ietf-l3-unicast-topology" as follows:

o A new topology type for an OSPF topology is introduced.

o Additional topology attributes are defined in a new grouping which
   augments l3-topology-attributes of the ietf-l3-unicast-topology
   module. The attributes include an OSPF area-id identifying the
   OSPF area.

      o Additional data objects for nodes are introduced by augmenting the
         l3-node-attributes of the l3-unicast-topology module. New objects
         include router-type and dr-interface-id for pseudonodes.

      o Links are augmented with ospf link attributes.

      In addition, the module extends notifications for events concerning
      Layer 3 nodes and links with OSPF attributes.

      It should be noted that the model defined here represents topology
      and is intended as an example. It does not define how to configure
      OSPF routers or interfaces.

B.1.2. OSPF Topology YANG Module

      The OSPF Topology YANG Module is specified below. As mentioned, the
      module is intended as an example for how the Layer 3 Unicast topology
      model can be extended to cover OSFP topologies, but it is not
      normative. Accordingly, the module is not delimited with CODE BEGINS
      and CODE ENDS tags.

   file "example-ospf-topology@2017-12-16.yang"
   module example-ospf-topology {
       yang-version 1.1;
       namespace "urn:example:example-ospf-topology";
       prefix "ex-ospft";
       import ietf-yang-types {
           prefix "yang";
       }
       import ietf-network {
           prefix "nw";
       }
       import ietf-network-topology {
           prefix "nt";
       }
       import ietf-l3-unicast-topology {
           prefix "l3t";
       }
       description
          "This module is intended as an example for how the
           Layer 3 Unicast topology model can be extended to cover
           OSFP topologies.";
       typedef area-id-type {
           type yang:dotted-quad;
           description
               "Area ID type.";
       }
       grouping ospf-topology-type {

```
        description
            "Identifies the OSPF topology type.";
        container ospf {
            presence "indicates OSPF Topology";
            description
                "Its presence identifies the OSPF topology type.";
        }
    }
    augment "/nw:networks/nw:network/nw:network-types/"
    + "l3t:l3-unicast-topology" {
        description
            "Defines the OSPF topology type.";
        uses ospf-topology-type;
    }
    augment "/nw:networks/nw:network/l3t:l3-topology-attributes" {
        when "../nw:network-types/l3t:l3-unicast-topology/" +
            "ex-ospft:ospf" {
            description
                "Augment only for OSPF topology";
        }
        description
            "Augment topology configuration";
        container ospf-topology-attributes {
            description
                "Containing topology attributes";
            leaf area-id {
                type area-id-type;
                description
                    "OSPF area ID";
            }
        }
    }
    augment "/nw:networks/nw:network/nw:node/l3t:l3-node-attributes" {
        when "../../nw:network-types/l3t:l3-unicast-topology/" +
            "ex-ospft:ospf" {
            description
                "Augment only for OSPF topology";
        }
        description
            "Augment node configuration";
        uses ospf-node-attributes;
    }
    augment "/nw:networks/nw:network/nt:link/l3t:l3-link-attributes" {
        when "../../nw:network-types/l3t:l3-unicast-topology/" +
            "ex-ospft:ospf" {
            description
                "Augment only for OSPF topology";
        }
```

```
            description
                "Augment link configuration";
            uses ospf-link-attributes;
        }
        grouping ospf-node-attributes {
            description
                "OSPF node scope attributes";
            container ospf-node-attributes {
                description
                    "Containing node attributes";
                choice router-type {
                    description
                        "Indicates router type";
                    case abr {
                        leaf abr {
                            type empty;
                            description
                                "The node is ABR";
                        }
                    }
                    case asbr {
                        leaf asbr {
                            type empty;
                            description
                                "The node is ASBR";
                        }
                    }
                    case internal {
                        leaf internal {
                            type empty;
                            description
                                "The node is internal";
                        }
                    }
                    case pseudonode {
                        leaf pseudonode {
                            type empty;
                            description
                                "The node is pseudonode";
                        }
                    }
                }
                leaf dr-interface-id {
                    when "../pseudonode" {
                        description
                            "Valid only for pseudonode";
                    }
                    type uint32;
```

```
                    default "0";
                    description
                        "For pseudonodes, DR interface-id";
                }
            }
        }
        grouping ospf-link-attributes {
            description
                "OSPF link scope attributes";
            container ospf-link-attributes {
                description
                    "Containing OSPF link attributes";
            }
        } // ospf-link-attributes
        augment "/l3t:l3-node-event" {
            description
                "OSPF node event";
            uses ospf-topology-type;
            uses ospf-node-attributes;
        }
        augment "/l3t:l3-link-event" {
            description
                "OSPF link event";
            uses ospf-topology-type;
            uses ospf-link-attributes;
        }
    }
```

Appendix C. An Example

   This section contains an example of an instance data tree in JSON
   encoding [RFC7951]. The example instantiates ietf-l3-unicast-
   topology for the topology that is depicted in the following diagram.
   There are three nodes, D1, D2, and D3. D1 has three termination
   points, 1-0-1, 1-2-1, and 1-3-1. D2 has three termination points as
   well, 2-1-1, 2-0-1, and 2-3-1. D3 has two termination points, 3-1-1
   and 3-2-1. In addition there are six links, two between each pair of
   nodes with one going in each direction.

```
            +-----------+                 +-----------+
            |    D1     |                 |    D2     |
            /-\       /-\-----------------\/-\       /-\
            | | 1-0-1 | |---------------->| | 2-1-1 | |
            | |   1-2-1 | |<---------------| |   2-0-1 | |
            \-/  1-3-1  \-/                 \-/  2-3-1  \-/
             |  /----\   |                   |  /----\   |
            +---|    |---+                 +---|    |---+
                \----/                         \----/
              A  |                           A  |
              |  |                           |  |
              |  |                           |  |
              |  |        +-----------+      |  |
              |  |        |    D3     |      |  |
              |  |        /-\       /-\      |  |
              |  +----->| | 3-1-1 | |------+  |
            +---------| |   3-2-1 | |<---------+
                       \-/         \-/
                        |           |
                       +-----------+
```
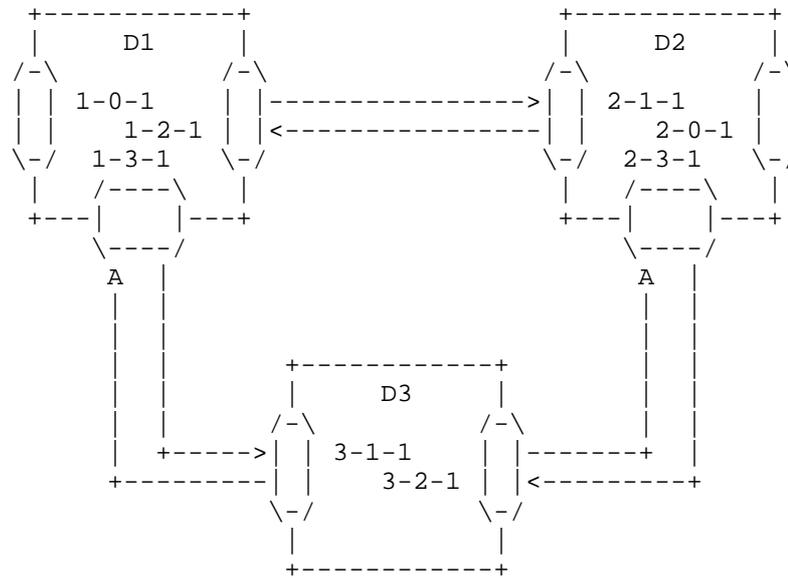
                    Figure 2: A network topology example

   The corresponding instance data tree is depicted below:

```
{
  "ietf-network:networks": {
    "network": [
      {
        "network-types": {
          "ietf-l3-unicast-topology:l3-unicast-topology": {}
        },
        "network-id": "l3-topo-example",
        "node": [
          {
            "node-id": "D1",
            "termination-point": [
              {
                "tp-id": "1-0-1",
                "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                  "unnumbered-id:": 101
                }
              },
              {
                "tp-id": "1-2-1",
                "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                  "unnumbered-id:": 121
```

```
              }
            },
            {
              "tp-id": "1-3-1",
              "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                "unnumbered-id:": 131
              }
            }
          ],
          "ietf-l3-unicast-topology:l3-node-attributes": {
            "router-id": ["203.0.113.1"]
          }
        },
        {
          "node-id": "D2",
          "termination-point": [
            {
              "tp-id": "2-0-1",
              "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                "unnumbered-id:": 201
              }
            },
            {
              "tp-id": "2-1-1",
              "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                "unnumbered-id:": 211
              }
            },
            {
              "tp-id": "2-3-1",
              "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                "unnumbered-id:": 231
              }
            }
          ],
          "ietf-l3-unicast-topology:l3-node-attributes": {
            "router-id": ["203.0.113.2"]
          }
        },
        {
          "node-id": "D3",
          "termination-point": [
            {
              "tp-id": "3-1-1",
              "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                "unnumbered-id:": 311
              }
            },
```

```
              {
                "tp-id": "3-2-1",
                "ietf-l3-unicast-topology:l3-termination-point-attributes": {
                  "unnumbered-id:": 321
                }
              }
            ],
            "ietf-l3-unicast-topology:l3-node-attributes": {
              "router-id": ["203.0.113.3"]
            }
          }
        ],
        "ietf-network-topology:link": [
          {
            "link-id": "D1,1-2-1,D2,2-1-1",
            "destination": {
              "source-node": "D1",
              "source-tp": "1-2-1"
            }
            "destination": {
              "dest-node": "D2",
              "dest-tp": "2-1-1"
            },
            "ietf-l3-unicast-topology:l3-link-attributes": {
              "metric1": "100"
            }
          },
          {
            "link-id": "D2,2-1-1,D1,1-2-1",
            "destination": {
              "source-node": "D2",
              "source-tp": "2-1-1"
            }
            "destination": {
              "dest-node": "D1",
              "dest-tp": "1-2-1"
            },
            "ietf-l3-unicast-topology:l3-link-attributes": {
              "metric1": "100"
            }
          },
          {
            "link-id": "D1,1-3-1,D3,3-1-1",
            "destination": {
              "source-node": "D1",
              "source-tp": "1-3-1"
            }
            "destination": {
```

```
              "dest-node": "D3",
              "dest-tp": "3-1-1"
            },
            "ietf-l3-unicast-topology:l3-link-attributes": {
              "metric1": "100"
            }
          },
          {
            "link-id": "D3,3-1-1,D1,1-3-1",
            "destination": {
              "source-node": "D3",
              "source-tp": "3-1-1"
            }
            "destination": {
              "dest-node": "D1",
              "dest-tp": "1-3-1"
            },
            "ietf-l3-unicast-topology:l3-link-attributes": {
              "metric1": "100"
            }
          },
          {
            "link-id": "D2,2-3-1,D3,3-2-1",
            "destination": {
              "source-node": "D2",
              "source-tp": "2-3-1"
            }
            "destination": {
              "dest-node": "D3",
              "dest-tp": "3-2-1"
            },
            "ietf-l3-unicast-topology:l3-link-attributes": {
              "metric1": "100"
            }
          },
          {
            "link-id": "D3,3-2-1,D2,2-3-1",
            "destination": {
              "source-node": "D3",
              "source-tp": "3-2-1"
            }
            "destination": {
              "dest-node": "D2",
              "dest-tp": "2-3-1"
            },
            "ietf-l3-unicast-topology:l3-link-attributes": {
              "metric1": "100"
            }
```

```
            }
          ]
        }
      ]
    }
  }
}
```

                        Figure 3: Instance data tree

Authors' Addresses

   Alexander Clemm
   Huawei USA

   EMail: ludwig@clemm.org

   Jan Medved
   Cisco

   EMail: jmedved@cisco.com

   Robert Varga
   Pantheon Technologies SRO

   EMail: robert.varga@pantheon.tech

   Xufeng Liu
   Jabil

   EMail: Xufeng_Liu@jabil.com

   Hariharan Ananthakrishnan
   Packet Design

   EMail: hari@packetdesign.com

   Nitin Bahadur
   Bracket Computing

   EMail: nitin_bahadur@yahoo.com

Network Working Group   A. Clemm
Internet-Draft   Huawei
Intended status: Standards Track   J. Medved
Expires: June 21, 2018   Cisco
                                                               R. Varga
                                              Pantheon Technologies SRO
                                                             N. Bahadur
                                                       Bracket Computing
                                                    H. Ananthakrishnan
                                                          Packet Design
                                                                X. Liu
                                                                 Jabil
                                                     December 18, 2017

                     A Data Model for Network Topologies
                    draft-ietf-i2rs-yang-network-topo-20.txt

Abstract

   This document defines an abstract (generic) YANG data model for
   network/service topologies and inventories. The data model serves as
   a base model which is augmented with technology-specific details in
   other, more specific topology and inventory data models.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   This document introduces an abstract (base) YANG [RFC7950] data model
   [RFC3444] to represent networks and topologies. The data model is
   divided into two parts. The first part of the data model defines a
   network data model that enables the definition of network hierarchies
   (i.e. network stacks of networks that are layered on top of each
   other) and to maintain an inventory of nodes contained in a network.
   The second part of the data model augments the basic network data
   model with information to describe topology information.
   Specifically, it adds the concepts of links and termination points to
   describe how nodes in a network are connected to each other.
   Moreover the data model introduces vertical layering relationships
   between networks that can be augmented to cover both network
   inventories and network/service topologies.

   While it would be possible to combine both parts into a single data
   model, the separation facilitates integration of network topology and
   network inventory data models, because it allows to augment network
   inventory information separately and without concern for topology
   into the network data model.

   The data model can be augmented to describe the specifics of
   particular types of networks and topologies. For example, an
   augmenting data model can provide network node information with
   attributes that are specific to a particular network type. Examples
   of augmenting models include data models for Layer 2 network
   topologies, Layer 3 network topologies, such as Unicast IGP, IS-IS
   [RFC1195] and OSPF [RFC2328], traffic engineering (TE) data
   [RFC3209], or any of the variety of transport and service topologies.
   Information specific to particular network types will be captured in
   separate, technology-specific data models.

   The basic data models introduced in this document are generic in
   nature and can be applied to many network and service topologies and
   inventories. The data models allow applications to operate on an
   inventory or topology of any network at a generic level, where the
   specifics of particular inventory/topology types are not required.
   At the same time, where data specific to a network type does comes

into play and the data model is augmented, the instantiated data
still adheres to the same structure and is represented in a
consistent fashion. This also facilitates the representation of
network hierarchies and dependencies between different network
components and network types.

The abstract (base) network YANG module introduced in this document,
entitled "ietf-network.yang", contains a list of abstract network
nodes and defines the concept of network hierarchy (network stack).
The abstract network node can be augmented in inventory and topology
data models with inventory and topology specific attributes. Network
hierarchy (stack) allows any given network to have one or more
"supporting networks". The relationship of the base network data
model, the inventory data models and the topology data models is
shown in the following figure (dotted lines in the figure denote
possible augmentations to models defined in this document).

```
                +-----------------------+
                |                       |
                | Abstract Network Model|
                |                       |
                +-----------------------+
                            |
                   +-------+-------+
                   |               |
                   V               V
              +-----------+  ..............
              | Abstract  |  : Inventory  :
              | Topology  |  :  Model(s)  :
              |  Model    |  :            :
              +-----------+  ''''''''''''''
                    |
        +-----------+-----------+------------+
        |           |           |            |
        V           V           V            V
   ............  ............  ............  ............
   :   L1    :  :   L2    :  :   L3    :  : Service :
   : Topology:  : Topology:  : Topology:  : Topology:
   :  Model  :  :  Model  :  :  Model  :  :  Model  :
   ''''''''''''  ''''''''''''  ''''''''''''  ''''''''''''
```

Figure 1: The network data model structure

The network-topology YANG module introduced in this document,
entitled "ietf-network-topology.yang", defines a generic topology
data model at its most general level of abstraction. The module
defines a topology graph and components from which it is composed:
nodes, edges and termination points. Nodes (from the ietf-

network.yang module) represent graph vertices and links represent
graph edges. Nodes also contain termination points that anchor the
links. A network can contain multiple topologies, for example
topologies at different layers and overlay topologies. The data
model therefore allows to capture relationships between topologies,
as well as dependencies between nodes and termination points across
topologies. An example of a topology stack is shown in the following
figure.

```
       +-------------------------------------+
      /              _[X1]_        "Service"  /
     /            _/  :   \_                  /
    /          _/      :     \_              /
   /        _/          :       \_          /
  /        /             :         \        /
 /       [X2]_____[X3]      /
+---------:--------------:------:-------+
         :              :     :
     +----:--------------:----:--------------+
    /     :              :    :      "L3" /
   /      :              :    :          /
  /       :              : :            /
 /       [Y1]_____[Y2]          /
/         *               * *         /
/         *               * *        /
+--------------*-------------*--*-------+
              *            *   *
     +---------*----------*----*-------------+
    /     [Z1]_____[Z1] "Optical" /
   /        \_         *   _/             /
  /          \_        *  _/             /
 /            \_      * _/              /
/              \  *  /                 /
/              [Z]                    /
+-------------------------------------+
```
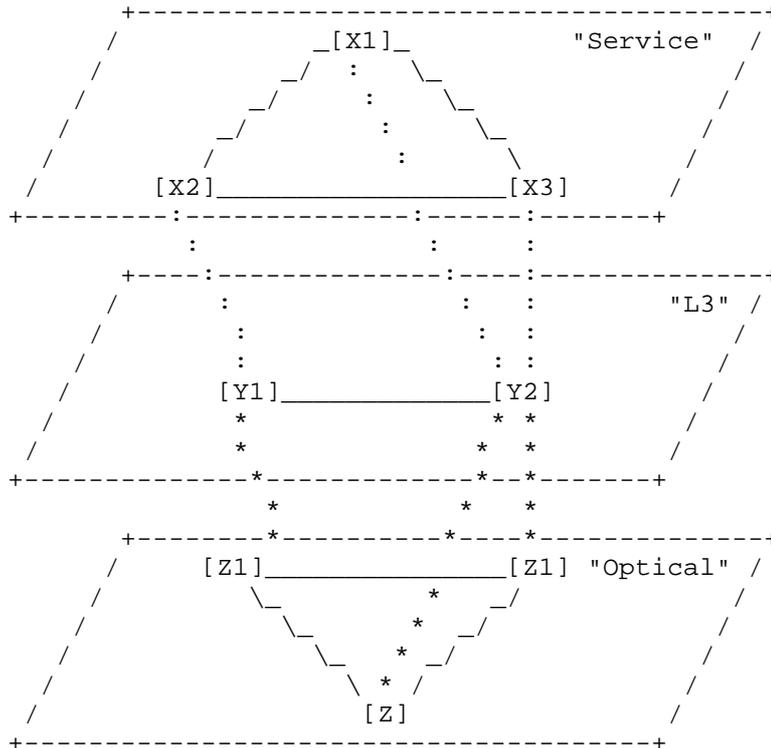
              Figure 2: Topology hierarchy (stack) example

   The figure shows three topology levels. At top, the "Service"
   topology shows relationships between service entities, such as
   service functions in a service chain. The "L3" topology shows
   network elements at Layer 3 (IP) and the "Optical" topology shows
   network elements at Layer 1. Service functions in the "Service"
   topology are mapped onto network elements in the "L3" topology, which
   in turn are mapped onto network elements in the "Optical" topology.
   The figure shows two Service Functions (X1 and X3) mapping onto a
   single L3 network element (Y2); this could happen, for example, if
   two service functions reside in the same VM (or server) and share the

same set of network interfaces. The figure shows a single "L3"
network element (Y2) mapped onto multiple "Optical" network elements
(Z and Z1). This could happen, for example, if a single IP router
attaches to multiple Reconfigurable Optical Add/Drop Multiplexers
(ROADMs) in the optical domain.

Another example of a service topology stack is shown in the following
figure.

```
                          VPN1                        VPN2
        +--------------------+     +--------------------+
       /   [Y5]...          /     /  [Z5]_____[Z3]    /
      /    /  \  :         /     /   : \_      / :    /
     /    /    \  :       /     /    :   \_   /  :   /
    /    /      \  :     /     /     :     \ /   :  /
   /   [Y4]____[Y1] :   /     /      :    [Z2]   : /
  +------:-------:---:--+     +---:---------:-----:-+
         :       :   :           :         :     :
         :       :   :           :         :     :
         :   +-------:---:-----:-----------:-----:-----+
         : /    [X1]__:___:_____[X2]   :     /
         :/      / \_  : :        _____/ /   :    /
         :      /    \_ :   _____/      /    :   /
        /:     /      \: /            /     :  /
       / :    /       [X5]           /      : /
      /   :  /      __/ \__         /      : /
     /     : /    ___/      \__    /      : /
    /      :/ ___/             \  /      : /
   /     [X4]_____[X3]..:     /
  +----------------------------------------+
                   L3 Topology
```
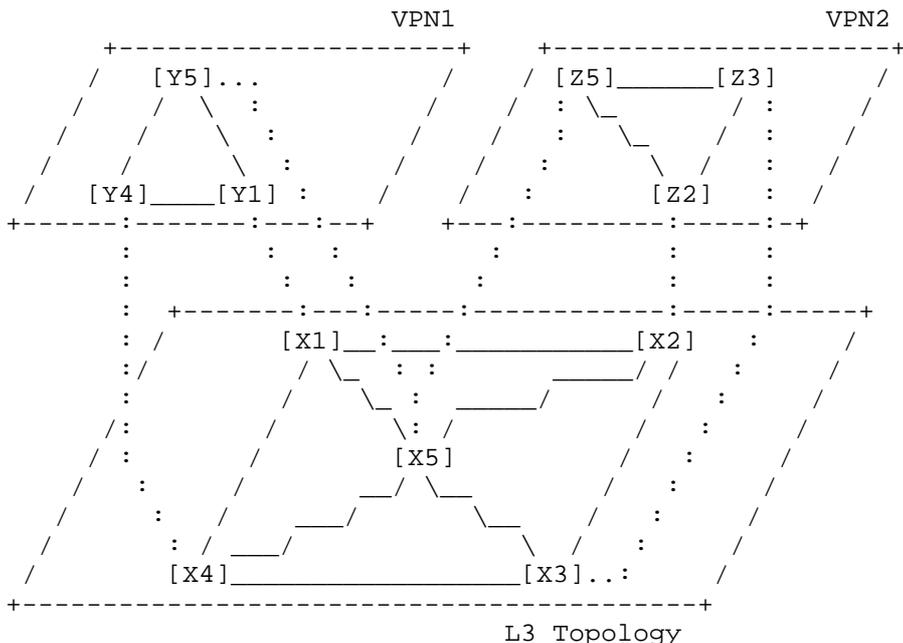
Figure 3: Topology hierarchy (stack) example

The figure shows two VPN service topologies (VPN1 and VPN2)
instantiated over a common L3 topology. Each VPN service topology is
mapped onto a subset of nodes from the common L3 topology.

There are multiple applications for such a data model. For example,
within the context of I2RS, nodes within the network can use the data
model to capture their understanding of the overall network topology
and expose it to a network controller. A network controller can then
use the instantiated topology data to compare and reconcile its own
view of the network topology with that of the network elements that
it controls. Alternatively, nodes within the network could propagate
this understanding to compare and reconcile this understanding either
among themselves or with help of a controller. Beyond the network
element and the immediate context of I2RS itself, a network

controller might even use the data model to represent its view of the
topology that it controls and expose it to applications north of
itself. Further use cases that the data model can be applied to are
described in [I-D.draft-ietf-i2rs-usecase-reqs-summary].

In this data model, a network is categorized as either system
controlled or not. If a network is system controlled, then it is
automatically populated by the server and represents dynamically
learned information that can be read from the operational state
datastore. The data model can also be used to create or modify
network topologies that might be associated with an inventory model
or with an overlay network. Such a network is not system controlled
but configured by a client.

The data model allows a network to refer to a supporting-network,
supporting-nodes, supporting-links, etc. The data model also allows
to layer a network that is configured on top of one that is system
controlled. This permits the configuration of overlay networks on
top of networks that are discovered. Specifically, this data model
is structured to support being implemented as part of the ephemeral
datastore [I-D.draft-ietf-netmod-revised-datastores], defined as
requirement Ephemeral-REQ-03 in [RFC8242]. This allows network
topology data that is written, i.e. configured by a client and not
system controlled, to refer to a dynamically learned data that is
controlled by the system, not configured by a client. A simple use
case might involve creating an overlay network that is supported by
the dynamically discovered IP routed network topology. When an
implementation places written data for this data model in the
ephemeral data store, then such a network MAY refer to another
network that is system controlled.

2. Key Words

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.

3. Definitions and Acronyms

Datastore: A conceptual place to store and access information. A
datastore might be implemented, for example, using files, a database,
flash memory locations, or combinations thereof. A datastore maps to
an instantiated YANG data tree. (Definition adopted from
[I-D.draft-ietf-netmod-revised-datastores])

Data subtree: An instantiated data node and the data nodes that are
hierarchically contained within it.

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

OSPF: Open Shortest Path First, a link state routing protocol

URI: Uniform Resource Identifier

4. Model Structure Details

4.1. Base Network Model

The abstract (base) network data model is defined in the ietf-
network.yang module. Its structure is shown in the following figure.
The notation syntax follows
[I-D.draft-ietf-netmod-yang-tree-diagrams].

```
module: ietf-network
+--rw networks
   +--rw network* [network-id]
      +--rw network-id            network-id
      +--rw network-types
      +--rw supporting-network* [network-ref]
      |  +--rw network-ref    -> /networks/network/network-id
      +--rw node* [node-id]
         +--rw node-id             node-id
         +--rw supporting-node* [network-ref node-ref]
            +--rw network-ref     -> ../../../supporting-network/ +
            |                        network-ref
            +--rw node-ref        -> /networks/network/node/node-id
```

Figure 4: The structure of the abstract (base) network data model

The data model contains a container with a list of networks. Each
network is captured in its own list entry, distinguished via a
network-id.

A network has a certain type, such as L2, L3, OSPF or IS-IS. A
network can even have multiple types simultaneously. The type, or
types, are captured underneath the container "network-types". In
this module it serves merely as an augmentation target; network-
specific modules will later introduce new data nodes to represent new

network types below this target, i.e. insert them below "network-
types" by ways of YANG augmentation.

When a network is of a certain type, it will contain a corresponding
data node. Network types SHOULD always be represented using presence
containers, not leafs of empty type. This allows the representation
of hierarchies of network subtypes within the instance information.
For example, an instance of an OSPF network (which, at the same time,
is a layer 3 unicast IGP network) would contain underneath "network-
types" another presence container "l3-unicast-igp-network", which in
turn would contain a presence container "ospf-network". Actual
examples of this pattern can be found in
[I-D.draft-ietf-i2rs-yang-l3-topology].

A network can in turn be part of a hierarchy of networks, building on
top of other networks. Any such networks are captured in the list
"supporting-network". A supporting network is in effect an underlay
network.

Furthermore, a network contains an inventory of nodes that are part
of the network. The nodes of a network are captured in their own
list. Each node is identified relative to its containing network by
a node-id.

It should be noted that a node does not exist independently of a
network; instead it is a part of the network that it is contained in.
In cases where the same device or entity takes part in multiple
networks, or at multiple layers of a networking stack, the same
device or entity will be represented by multiple nodes, one for each
network. In other words, the node represents an abstraction of the
device for the particular network that it a is part of. To represent
that the same entity or same device is part of multiple topologies or
networks, it is possible to create one "physical" network with a list
of nodes for each of the devices or entities. This (physical)
network, respectively the (entities) nodes in that network, can then
be referred to as underlay network and nodes from the other (logical)
networks and nodes, respectively. Note that the data model allows
for the definition of more than one underlay network (and node),
allowing for simultaneous representation of layered network and
service topologies and their physical instantiation.

Similar to a network, a node can be supported by other nodes, and map
onto one or more other nodes in an underlay network. This is
captured in the list "supporting-node". The resulting hierarchy of
nodes allows also for the representation of device stacks, where a
node at one level is supported by a set of nodes at an underlying
level. For example, a "router" node might be supported by a node
representing a route processor and separate nodes for various line

cards and service modules, a virtual router might be supported or
hosted on a physical device represented by a separate node, and so
on.

Network data of a network at a particular layer can come into being
in one of two ways. In one way, network data is configured by client
applications, for example in case of overlay networks that are
configured by an SDN Controller application. In another way, it is
automatically controlled by the system, in case of networks that can
be discovered. It is possible for a configured (overlay) network to
refer to a (discovered) underlay network.

The revised datastore architecture
[I-D.draft-ietf-netmod-revised-datastores] is used to account for
those possibilities. Specifically, for each network, the origin of
its data is indicated per the "origin" metadata annotation -
"intended" for data that was configured by a client application,
"learned" for data that is discovered. Network data that is
discovered is automatically populated as part of the operational
state datastore. Network data that is configured is part of the
configuration and intended datastores, respectively. Configured
network data that is actually in effect is in addition reflected in
the operational state datastore. Data in the operational state
datastore will always have complete referential integrity. Should a
configured data item (such as a node) have a dangling reference that
refers to a non-existing data item (such as a supporting node), the
configured data item will automatically be removed from the
operational state datastore and thus only appear in the intended
datastore. It will be up to the client application (such as an SDN
controller) to resolve the situation and ensure that the reference to
the supporting resources is configured properly.

4.2. Base Network Topology Data Model

The abstract (base) network topology data model is defined in the
"ietf-network-topology.yang" module. It builds on the network data
model defined in the "ietf-network.yang" module, augmenting it with
links (defining how nodes are connected) and termination-points
(which anchor the links and are contained in nodes). The structure
of the network topology module is shown in the following figure. The
notation syntax follows [I-D.draft-ietf-netmod-yang-tree-diagrams].

```
module: ietf-network-topology
augment /nw:networks/nw:network:
   +--rw link* [link-id]
      +--rw link-id            link-id
      +--rw source
      |  +--rw source-node?   -> ../../../nw:node/node-id
      |  +--rw source-tp?     -> ../../../nw:node[nw:node-id=current()/+
      |                          ../source-node]/termination-point/tp-id
      +--rw destination
      |  +--rw dest-node?   -> ../../../nw:node/node-id
      |  +--rw dest-tp?     -> ../../../nw:node[nw:node-id=current()/+
      |                        ../dest-node]/termination-point/tp-id
      +--rw supporting-link* [network-ref link-ref]
         +--rw network-ref     -> ../../../nw:supporting-network/+
         |                        network-ref
         +--rw link-ref        -> /nw:networks/network+
                                  [nw:network-id=current()/../network-ref]/+
                                  link/link-id
augment /nw:networks/nw:network/nw:node:
   +--rw termination-point* [tp-id]
      +--rw tp-id                         tp-id
      +--rw supporting-termination-point* [network-ref node-ref tp-ref]
         +--rw network-ref     -> ../../../nw:supporting-node/network-ref
         +--rw node-ref        -> ../../../nw:supporting-node/node-ref
         +--rw tp-ref          -> /nw:networks/network[nw:network-id=+
                                  current()/../network-ref]/node+
                                  [nw:node-id=current()/../node-ref]/+
                                  termination-point/tp-id
```

         Figure 5: The structure of the abstract (base) network topology data
                                 model

   A node has a list of termination points that are used to terminate
   links. An example of a termination point might be a physical or
   logical port or, more generally, an interface.

   Like a node, a termination point can in turn be supported by an
   underlying termination point, contained in the supporting node of the
   underlay network.

   A link is identified by a link-id that uniquely identifies the link
   within a given topology. Links are point-to-point and
   unidirectional. Accordingly, a link contains a source and a
   destination. Both source and destination reference a corresponding
   node, as well as a termination point on that node. Similar to a
   node, a link can map onto one or more links in an underlay topology
   (which are terminated by the corresponding underlay termination
   points). This is captured in the list "supporting-link".

4.3. Extending the data model

   In order to derive a data model for a specific type of network, the
   base data model can be extended. This can be done roughly as
   follows: for the new network type, a new YANG module is introduced.
   In this module, a number of augmentations are defined against the
   network and network-topology YANG modules.

   We start with augmentations against the ietf-network.yang module.
   First, a new network type needs to be defined. For this, a presence
   container that represents the new network type is defined. It is
   inserted by means of augmentation below the network-types container.
   Subsequently, data nodes for any network-type specific node
   parameters are defined and augmented into the node list. The new
   data nodes can be defined as conditional ("when") on the presence of
   the corresponding network type in the containing network. In cases
   where there are any requirements or restrictions in terms of network
   hierarchies, such as when a network of a new network-type requires a
   specific type of underlay network, it is possible to define
   corresponding constraints as well and augment the supporting-network
   list accordingly. However, care should be taken to avoid excessive
   definitions of constraints.

   Subsequently, augmentations are defined against ietf-network-
   topology.yang. Data nodes are defined both for link parameters, as
   well as termination point parameters, that are specific to the new
   network type. Those data nodes are inserted by way of augmentation
   into the link and termination-point lists, respectively. Again, data
   nodes can be defined as conditional on the presence of the
   corresponding network-type in the containing network, by adding a
   corresponding "when"-statement.

   It is possible, but not required, to group data nodes for a given
   network-type under a dedicated container. Doing so introduces
   further structure, but lengthens data node path names.

   In cases where a hierarchy of network types is defined, augmentations
   can in turn be applied against augmenting modules, with the module of
   a more specific network type augmenting the module of a network of a
   more general type.

4.4. Discussion and selected design decisions

4.4.1. Container structure

   Rather than maintaining lists in separate containers, the data model
   is kept relatively flat in terms of its containment structure. Lists
   of nodes, links, termination-points, and supporting-nodes,

supporting-links, and supporting-termination-points are not kept in
separate containers. Therefore, path identifiers are used to refer
to specific nodes, be it in management operations or in
specifications of constraints, can remain relatively compact. Of
course, this means there is no separate structure in instance
information that separates elements of different lists from one
another. Such structure is semantically not required, although it
might enhance human readability in some cases.

4.4.2. Underlay hierarchies and mappings

   To minimize assumptions of what a particular entity might actually
   represent, mappings between networks, nodes, links, and termination
   points are kept strictly generic. For example, no assumptions are
   made whether a termination point actually refers to an interface, or
   whether a node refers to a specific "system" or device; the data
   model at this generic level makes no provisions for that.

   Where additional specifics about mappings between upper and lower
   layers are required, those can be captured in augmenting modules.
   For example, to express that a termination point in a particular
   network type maps to an interface, an augmenting module can introduce
   an augmentation to the termination point which introduces a leaf of
   type ifref that references the corresponding interface [RFC7223].
   Similarly, if a node maps to a particular device or network element,
   an augmenting module can augment the node data with a leaf that
   references the network element.

   It is possible for links at one level of a hierarchy to map to
   multiple links at another level of the hierarchy. For example, a VPN
   topology might model VPN tunnels as links. Where a VPN tunnel maps
   to a path that is composed of a chain of several links, the link will
   contain a list of those supporting links. Likewise, it is possible
   for a link at one level of a hierarchy to aggregate a bundle of links
   at another level of the hierarchy.

4.4.3. Dealing with changes in underlay networks

   It is possible for a network to undergo churn even as other networks
   are layered on top of it. When a supporting node, link, or
   termination point is deleted, the supporting leafrefs in the overlay
   will be left dangling. To allow for this possibility, the data model
   makes use of the "require-instance" construct of YANG 1.1 [RFC7950].

   A dangling leafref of a configured object leaves the corresponding
   instance in a state in which it lacks referential integrity,
   rendering it in effect inoperational. Any corresponding object
   instance is therefore removed from the operational state datastore

until the situation has been resolved, i.e. until either the
supporting object is added to the operational state datastore, or
until the instance is reconfigured to refer to another object that is
actually reflected in the operational state datastore. It does
remain part of the intended datastore.

It is the responsibility of the application maintaining the overlay
to deal with the possibility of churn in the underlay network. When
a server receives a request to configure an overlay network, it
SHOULD validate whether supporting nodes/links/tps refer to nodes in
the underlay are actually in existence, i.e. nodes which are
reflected in the operational state datastore. Configuration requests
in which supporting nodes/links/tps refer to objects currently not in
existence SHOULD be rejected. It is the responsibility of the
application to update the overlay when a supporting node/link/tp is
deleted at a later point in time. For this purpose, an application
might subscribe to updates when changes to the underlay occur, for
example using mechanisms defined in
[I-D.draft-ietf-netconf-yang-push].

4.4.4. Use of groupings

The data model makes use of groupings, instead of simply defining
data nodes "in-line". This makes it easier to include the
corresponding data nodes in notifications, which then do not need to
respecify each data node that is to be included. The tradeoff for
this is that it makes the specification of constraints more complex,
because constraints involving data nodes outside the grouping need to
be specified in conjunction with a "uses" statement where the
grouping is applied. This also means that constraints and XPath-
statements need to be specified in such a way that they navigate
"down" first and select entire sets of nodes, as opposed to being
able to simply specify them against individual data nodes.

4.4.5. Cardinality and directionality of links

The topology data model includes links that are point-to-point and
unidirectional. It does not directly support multipoint and
bidirectional links. While this may appear as a limitation, it does
keep the data model simple, generic, and allows it to very easily be
subjected to applications that make use of graph algorithms. Bi-
directional connections can be represented through pairs of
unidirectional links. Multipoint networks can be represented through
pseudo-nodes (similar to IS-IS, for example). By introducing
hierarchies of nodes, with nodes at one level mapping onto a set of
other nodes at another level, and introducing new links for nodes at
that level, topologies with connections representing non-point-to-
point communication patterns can be represented.

4.4.6. Multihoming and link aggregation

   Links are terminated by a single termination point, not sets of
   termination points. Connections involving multihoming or link
   aggregation schemes need to be represented using multiple point-to-
   point links, then defining a link at a higher layer that is supported
   by those individual links.

4.4.7. Mapping redundancy

   In a hierarchy of networks, there are nodes mapping to nodes, links
   mapping to links, and termination points mapping to termination
   points. Some of this information is redundant. Specifically, if the
   link-to-links mapping is known, and the termination points of each
   link are known, termination point mapping information can be derived
   via transitive closure and does not have to be explicitly configured.
   Nonetheless, in order to not constrain applications regarding which
   mappings they want to configure and which should be derived, the data
   model does provide for the option to configure this information
   explicitly. The data model includes integrity constraints to allow
   for validating for consistency.

4.4.8. Typing

   A network's network types are represented using a container which
   contains a data node for each of its network types. A network can
   encompass several types of network simultaneously, hence a container
   is used instead of a case construct, with each network type in turn
   represented by a dedicated presence container itself. The reason for
   not simply using an empty leaf, or even simpler, do away even with
   the network container and just use a leaf-list of network-type
   instead, is to be able to represent "class hierarchies" of network
   types, with one network type refining the other. Network-type
   specific containers are to be defined in the network-specific
   modules, augmenting the network-types container.

4.4.9. Representing the same device in multiple networks

   One common requirement concerns the ability to represent that the
   same device can be part of multiple networks and topologies.
   However, the data model defines a node as relative to the network
   that it is contained in. The same node cannot be part of multiple
   topologies. In many cases, a node will be the abstraction of a
   particular device in a network. To reflect that the same device is
   part of multiple topologies, the following approach might be chosen:
   A new type of network to represent a "physical" (or "device") network
   is introduced, with nodes representing devices. This network forms

an underlay network for logical networks above it, with nodes of the
logical network mapping onto nodes in the physical network.

This scenario is depicted in the following figure. It depicts three
networks with two nodes each. A physical network P consists of an
inventory of two nodes, D1 and D2, each representing a device. A
second network, X, has a third network, Y, as its underlay. Both X
and Y also have the physical network P as underlay. X1 has both Y1
and D1 as underlay nodes, while Y1 has D1 as underlay node.
Likewise, X2 has both Y2 and D2 as underlay nodes, while Y2 has D2 as
underlay node. The fact that X1 and Y1 are both instantiated on the
same physical node D1 can be easily derived.

```
                    +--------------------+
                   /   [X1]____[X2]      /  X(Service Overlay)
                  +----:--:----:--------+
                   ..:     :..: :
                ........:     ....: : :....
          +-----:------------:--+     :      :...
         /   [Y1]____[Y2]....:  /       :..        :
        +------|-------|-------+        :..     :...
        Y(L3)  |       +--------------------:-----+ :
               |            +----:----|-:----------+
        +-----------------------/---[D1]  [D2]         /
                    +--------------------+
                    P (Physical network)
```
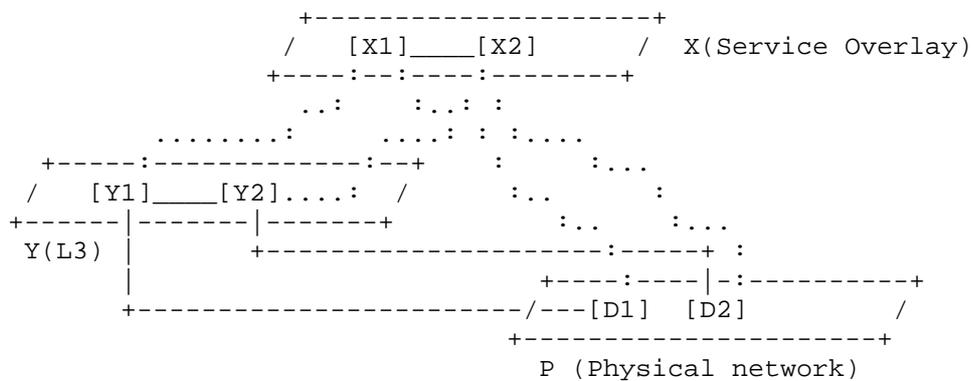
            Figure 6: Topology hierarchy example - multiple underlays

In the case of a physical network, nodes represent physical devices
and termination points physical ports. It should be noted that it is
also possible to augment the data model for a physical network-type,
defining augmentations that have nodes reference system information
and termination points reference physical interfaces, in order to
provide a bridge between network and device models.

4.4.10. Supporting client-configured and system-controlled network
         topology

YANG requires data nodes to be designated as either configuration
("config true") or operational data ("config false"), but not both,
yet it is important to have all network information, including
vertical cross-network dependencies, captured in one coherent data
model. In most cases, network topology information is discovered
about a network; the topology is considered a property of the network
that is reflected in the data model. That said, certain types of

topology need to also be configurable by an application, such as in
the case of overlay topologies.

The YANG data model for network topology designates all data as
"config true". The distinction between data that is actually
configured and data that is in effect, including data that is
discovered about the network, is provided through the datastores
introduced as part of the Network Management Datastore Architecture,
NMDA [I-D.draft-ietf-netmod-revised-datastores]. Network topology
data that is discovered is automatically populated as part of the
operational state datastore, <operational>. It is "system
controlled". Network topology that is configured is instantiated as
part of a configuration datastore, e.g. <intended>. Only when it has
actually taken effect, it is also instantiated as part of the
operational state datastore, i.e. <operational>.

Configured network topology will in general refer to an underlay
topology and include layering information, such as the supporting
node(s) underlying a node, supporting link(s) underlying a link, and
supporting termination point(s) underlying a termination point. The
supporting objects must be instantiated in the operational state
datastore in order for the dependent overlay object to be reflected
in the operational state datastore. Should a configured data item
(such as a node) have a dangling reference that refers to a non-
existing data item (such as a supporting node), the configured data
item will automatically be removed from <operational> and show up
only in <intended>. It will be up to the client application to
resolve the situation and ensure that the reference to the supporting
resources is configured properly.

For each network, the origin of its data is indicated per the
"origin" metadata [RFC7952] annotation defined in
[I-D.draft-ietf-netmod-revised-datastores]. In general, the origin
of discovered network data is "learned"; the origin of configured
network data is "intended".

4.4.11. Identifiers of string or URI type

The current data model defines identifiers of nodes, networks, links,
and termination points as URIs. An alternative would define them as
strings.

The case for strings is that they will be easier to implement. The
reason for choosing URIs is that the topology/node/tp exists in a
larger context, hence it is useful to be able to correlate
identifiers across systems. While strings, being the universal data
type, are easier for human beings, they also muddle things. What
typically happens is that strings have some structure which is

magically assigned and the knowledge of this structure has to be
communicated to each system working with the data. A URI makes the
structure explicit and also attaches additional semantics: the URI,
unlike a free-form string, can be fed into a URI resolver, which can
point to additional resources associated with the URI. This property
is important when the topology data is integrated into a larger, more
complex system.

5. Interactions with Other YANG Modules

The data model makes use of data types that have been defined in
[RFC6991].

This is a protocol independent YANG data model with topology
information. It is separate from and not linked with data models
that are used to configure routing protocols or routing information.
This includes e.g. data model "ietf-routing" [RFC8022].

The data model obeys the requirements for the ephemeral state found
in the document [RFC8242]. For ephemeral topology data that is
system controlled, the process tasked with maintaining topology
information will load information from the routing process (such as
OSPF) into the operational state datastore without relying on a
configuration datastore.

6. YANG Modules

6.1. Defining the Abstract Network: ietf-network.yang

NOTE TO RFC EDITOR: Please change the date in the file name after the
CODE BEGINS statement to the date of publication when published.

```
<CODE BEGINS> file "ietf-network@2017-12-18.yang"
module ietf-network {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network";
  prefix nw;

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

  organization
    "IETF I2RS (Interface to the Routing System) Working Group";

  contact
    "WG Web:    <http://tools.ietf.org/wg/i2rs/>
```

```
       WG List:    <mailto:i2rs@ietf.org>

       Editor:     Alexander Clemm
                   <mailto:ludwig@clemm.org>

       Editor:     Jan Medved
                   <mailto:jmedved@cisco.com>

       Editor:     Robert Varga
                   <mailto:robert.varga@pantheon.tech>

       Editor:     Nitin Bahadur
                   <mailto:nitin_bahadur@yahoo.com>

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

       Editor:     Xufeng Liu
                   <mailto:Xufeng_Liu@jabil.com>";

    description
      "This module defines a common base data model for a collection
       of nodes in a network. Node definitions are further used
       in network topologies and inventories.

       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
       draft-ietf-i2rs-yang-network-topo-20;
       see the RFC itself for full legal notices.

       NOTE TO RFC EDITOR: Please replace above reference to
       draft-ietf-i2rs-yang-network-topo-20 with RFC
       number when published (i.e. RFC xxxx).";

    revision 2017-12-18 {
      description
        "Initial revision.
         NOTE TO RFC EDITOR:
         (1) Please replace the following reference
```

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

       typedef node-id {
         type inet:uri;
         description
           "Identifier for a node.  The precise structure of the node-id
            will be up to the implementation.  Some implementations MAY
            for example, pick a uri that includes the network-id as
            part of the path. The identifier SHOULD be chosen such that
            the same node in a real network topology will always be
            identified through the same identifier, even if the data model
            is instantiated in separate datastores. An implementation MAY
            choose to capture semantics in the identifier, for example to
            indicate the type of node.";
       }

       typedef network-id {
         type inet:uri;
         description
           "Identifier for a network.  The precise structure of the
           network-id will be up to an implementation.
           The identifier SHOULD be chosen such that the same network
           will always be identified through the same identifier,
           even if the data model is instantiated in separate datastores.
           An implementation MAY choose to capture semantics in the
           identifier, for example to indicate the type of network.";
       }

       grouping network-ref {
         description
           "Contains the information necessary to reference a network,
            for example an underlay network.";
         leaf network-ref {
           type leafref {
             path "/nw:networks/nw:network/nw:network-id";
           require-instance false;
           }
           description
             "Used to reference a network, for example an underlay
              network.";
         }
       }
```

```
     grouping node-ref {
       description
         "Contains the information necessary to reference a node.";
       leaf node-ref {
         type leafref {
           path "/nw:networks/nw:network[nw:network-id=current()/../"+
             "network-ref]/nw:node/nw:node-id";
           require-instance false;
         }
         description
           "Used to reference a node.
            Nodes are identified relative to the network they are
            contained in.";
       }
       uses network-ref;
     }

     container networks {
       description
         "Serves as top-level container for a list of networks.";
       list network {
         key "network-id";
         description
           "Describes a network.
            A network typically contains an inventory of nodes,
            topological information (augmented through
            network-topology data model), as well as layering
            information.";
         leaf network-id {
           type network-id;
           description
             "Identifies a network.";
         }
         container network-types {
           description
             "Serves as an augmentation target.
              The network type is indicated through corresponding
              presence containers augmented into this container.";
         }
         list supporting-network {
           key "network-ref";
           description
             "An underlay network, used to represent layered network
              topologies.";
           leaf network-ref {
             type leafref {
               path "/nw:networks/nw:network/nw:network-id";
             require-instance false;
```

```
              }
              description
                "References the underlay network.";
            }
          }
          list node {
            key "node-id";
            description
              "The inventory of nodes of this network.";
            leaf node-id {
              type node-id;
              description
                "Identifies a node uniquely within the containing
                 network.";
            }
            list supporting-node {
              key "network-ref node-ref";
              description
                "Represents another node, in an underlay network, that
                 this node is supported by.  Used to represent layering
                 structure.";
              leaf network-ref {
                type leafref {
                  path "../../../nw:supporting-network/nw:network-ref";
                require-instance false;
                }
                description
                  "References the underlay network that the
                   underlay node is part of.";
              }
              leaf node-ref {
                type leafref {
                  path "/nw:networks/nw:network/nw:node/nw:node-id";
                require-instance false;
                }
                description
                  "References the underlay node itself.";
              }
            }
          }
        }
      }
    }

    <CODE ENDS>
```

6.2. Creating Abstract Network Topology: ietf-network-topology.yang

   NOTE TO RFC EDITOR: Please change the date in the file name after the
   CODE BEGINS statement to the date of publication when published.

```
 <CODE BEGINS> file "ietf-network-topology@2017-12-18.yang"
 module ietf-network-topology {
   yang-version 1.1;
   namespace "urn:ietf:params:xml:ns:yang:ietf-network-topology";
   prefix nt;

   import ietf-inet-types {
     prefix inet;
     reference
       "RFC 6991";
   }
   import ietf-network {
     prefix nw;
     reference
       "draft-ietf-i2rs-yang-network-topo-20
       NOTE TO RFC EDITOR:
       (1) Please replace above reference to
       draft-ietf-i2rs-yang-network-topo-20 with RFC
       number when published (i.e. RFC xxxx).
       (2) Please replace the date in the revision statement with the
        date of publication when published.";
   }

   organization
     "IETF I2RS (Interface to the Routing System) Working Group";

   contact
     "WG Web:    <http://tools.ietf.org/wg/i2rs/>
      WG List:   <mailto:i2rs@ietf.org>

      Editor:    Alexander Clemm
                 <mailto:ludwig@clemm.org>

      Editor:    Jan Medved
                 <mailto:jmedved@cisco.com>

      Editor:    Robert Varga
                 <mailto:robert.varga@pantheon.tech>

      Editor:    Nitin Bahadur
                 <mailto:nitin_bahadur@yahoo.com>

      Editor:    Hariharan Ananthakrishnan
```

```
                <mailto:hari@packetdesign.com>

     Editor:     Xufeng Liu
                 <mailto:Xufeng_Liu@jabil.com>";

   description
     "This module defines a common base model for network topology,
      augmenting the base network data model with links to connect
      nodes, as well as termination points to terminate links on nodes.

      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
      draft-ietf-i2rs-yang-network-topo-20;
      see the RFC itself for full legal notices.

      NOTE TO RFC EDITOR: Please replace above reference to
      draft-ietf-i2rs-yang-network-topo-20 with RFC
      number when published (i.e. RFC xxxx).";

   revision 2017-12-18 {
     description
       "Initial revision.
        NOTE TO RFC EDITOR: Please replace the following reference
        to draft-ietf-i2rs-yang-network-topo-20 with
        RFC number when published (i.e. RFC xxxx).";
     reference
       "draft-ietf-i2rs-yang-network-topo-20";
   }

   typedef link-id {
     type inet:uri;
     description
       "An identifier for a link in a topology.
        The precise structure of the link-id
        will be up to the implementation.
        The identifier SHOULD be chosen such that the same link in a
        real network topology will always be identified through the
        same identifier, even if the data model is instantiated in
            separate datastores. An implementation MAY choose to capture
```

```
      semantics in the identifier, for example to indicate the type
      of link and/or the type of topology that the link is a part
      of.";
}

typedef tp-id {
  type inet:uri;
  description
    "An identifier for termination points (TPs) on a node.
     The precise structure of the tp-id
     will be up to the implementation.
     The identifier SHOULD be chosen such that the same termination
     point in a real network topology will always be identified
     through the same identifier, even if the data model is
     instantiated in separate datastores. An implementation MAY
     choose to capture semantics in the identifier, for example to
     indicate the type of termination point and/or the type of node
     that contains the termination point.";
}

grouping link-ref {
  description
    "This grouping can be used to reference a link in a specific
     network.  While it is not used in this module, it is defined
     here for the convenience of augmenting modules.";
  leaf link-ref {
    type leafref {
      path "/nw:networks/nw:network[nw:network-id=current()/../"+
        "network-ref]/nt:link/nt:link-id";
      require-instance false;
    }
    description
      "A type for an absolute reference a link instance.
       (This type should not be used for relative references.
       In such a case, a relative path should be used instead.)";
  }
  uses nw:network-ref;
}

grouping tp-ref {
  description
    "This grouping can be used to references a termination point
     in a specific node.  While it is not used in this module, it
     is defined here for the convenience of augmenting modules.";
  leaf tp-ref {
    type leafref {
      path "/nw:networks/nw:network[nw:network-id=current()/../"+
        "network-ref]/nw:node[nw:node-id=current()/../"+
```

```
              "node-ref]/nt:termination-point/nt:tp-id";
            require-instance false;
          }
        description
          "A type for an absolute reference to a termination point.
           (This type should not be used for relative references.
           In such a case, a relative path should be used instead.)";
      }
      uses nw:node-ref;
    }

    augment "/nw:networks/nw:network" {
      description
        "Add links to the network data model.";
      list link {
        key "link-id";
        description
          "A network link connects a local (source) node and
           a remote (destination) node via a set of
           the respective node's termination points.
           It is possible to have several links between the same
           source and destination nodes.  Likewise, a link could
           potentially be re-homed between termination points.
           Therefore, in order to ensure that we would always know
           to distinguish between links, every link is identified by
           a dedicated link identifier.  Note that a link models a
           point-to-point link, not a multipoint link.";
        leaf link-id {
          type link-id;
          description
            "The identifier of a link in the topology.
             A link is specific to a topology to which it belongs.";
        }
        container source {
          description
            "This container holds the logical source of a particular
             link.";
          leaf source-node {
            type leafref {
              path "../../../nw:node/nw:node-id";
              require-instance false;
            }
            description
              "Source node identifier, must be in same topology.";
          }
          leaf source-tp {
            type leafref {
              path "../../../nw:node[nw:node-id=current()/../"+
```

```
              "source-node]/termination-point/tp-id";
            require-instance false;
          }
          description
            "Termination point within source node that terminates
             the link.";
        }
      }
      container destination {
        description
          "This container holds the logical destination of a
           particular link.";
        leaf dest-node {
          type leafref {
            path "../../../nw:node/nw:node-id";
          require-instance false;
          }
          description
            "Destination node identifier, must be in the same
             network.";
        }
        leaf dest-tp {
          type leafref {
            path "../../../nw:node[nw:node-id=current()/../"+
              "dest-node]/termination-point/tp-id";
            require-instance false;
          }
          description
            "Termination point within destination node that
             terminates the link.";
        }
      }
      list supporting-link {
        key "network-ref link-ref";
        description
          "Identifies the link, or links, that this link
           is dependent on.";
        leaf network-ref {
          type leafref {
            path "../../../nw:supporting-network/nw:network-ref";
          require-instance false;
          }
          description
            "This leaf identifies in which underlay topology
             the supporting link is present.";
        }
        leaf link-ref {
          type leafref {
```

```
            path "/nw:networks/nw:network[nw:network-id=current()/"+
              "../network-ref]/link/link-id";
            require-instance false;
          }
          description
            "This leaf identifies a link which is a part
             of this link's underlay. Reference loops in which
             a link identifies itself as its underlay, either
             directly or transitively, are not allowed.";
        }
      }
    }
  }
  augment "/nw:networks/nw:network/nw:node" {
    description
      "Augment termination points which terminate links.
       Termination points can ultimately be mapped to interfaces.";
    list termination-point {
      key "tp-id";
      description
        "A termination point can terminate a link.
         Depending on the type of topology, a termination point
         could, for example, refer to a port or an interface.";
      leaf tp-id {
        type tp-id;
        description
          "Termination point identifier.";
      }
      list supporting-termination-point {
        key "network-ref node-ref tp-ref";
        description
          "This list identifies any termination points that
           the termination point is dependent on, or maps onto.
           Those termination points will themselves be contained
           in a supporting node.
           This dependency information can be inferred from
           the dependencies between links.  For this reason,
           this item is not separately configurable.  Hence no
           corresponding constraint needs to be articulated.
           The corresponding information is simply provided by the
           implementing system.";
        leaf network-ref {
          type leafref {
            path "../../../nw:supporting-node/nw:network-ref";
          require-instance false;
          }
          description
            "This leaf identifies in which topology the
```

```
                    supporting termination point is present.";
            }
            leaf node-ref {
              type leafref {
                path "../../../nw:supporting-node/nw:node-ref";
              require-instance false;
              }
              description
                "This leaf identifies in which node the supporting
                 termination point is present.";
            }
            leaf tp-ref {
              type leafref {
                path "/nw:networks/nw:network[nw:network-id=current()/"+
                  "../network-ref]/nw:node[nw:node-id=current()/../"+
                  "node-ref]/termination-point/tp-id";
                require-instance false;
              }
              description
                "Reference to the underlay node, must be in a
                 different topology";
            }
          }
        }
      }
    }
  }
```

 <CODE ENDS>

7. IANA Considerations

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

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

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

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

   URI:urn:ietf:params:xml:ns:yang:ietf-network-topology-state
   Registrant Contact: The IESG.

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

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

   NOTE TO THE RFC EDITOR: In the list below, please replace references
   to "draft-ietf-i2rs-yang-network-topo-20 (RFC form)" with RFC number
   when published (i.e. RFC xxxx).

   Name: ietf-network
   Namespace: urn:ietf:params:xml:ns:yang:ietf-network
   Prefix: nw
   Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

   Name: ietf-network-topology
   Namespace: urn:ietf:params:xml:ns:yang:ietf-network-topology
   Prefix: nt
   Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

   Name: ietf-network-state
   Namespace: urn:ietf:params:xml:ns:yang:ietf-network-state
   Prefix: nw-s
   Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

   Name: ietf-network-topology-state
   Namespace: urn:ietf:params:xml:ns:yang:ietf-network-topology-state
   Prefix: nt-s
   Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

8. Security Considerations

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

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

   The network topology and inventory created by this module reveals
   information about the structure of networks that could be very
   helpful to an attacker. As a privacy consideration, while there is
   no personally identifiable information defined in this module, it is

possible that some node identifiers may be associated with devices that are in turn associated with specific users.

The YANG modules define information that can be configurable in certain instances, for example in the case of overlay topologies that can be created by client applications. In such cases, a malicious client could introduce topologies that are undesired. Specifically, a malicious client could attempt to remove or add a node, a link, a termination point, by creating or deleting corresponding elements in the node, link, and termination point lists, respectively. In the case of a topology that is learned, the server will automatically prohibit such misconfiguration attempts. In the case of a topology that is configured, i.e. whose origin is "intended", the undesired configuration could become effective and be reflected in the operational state datastore, leading to disruption of services provided via this topology might be disrupted. For example, the topology could be "cut" or be configured in a suboptimal way, leading to increased consumption of resources in the underlay network due to resulting routing and bandwidth utilization inefficiencies. Likewise, it could lead to degradation of service levels as well as possibly disruption of service. For those reasons, it is important that the NETCONF access control model is vigorously applied to prevent topology misconfiguration by unauthorized clients.

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

o network: A malicious client could attempt to remove or add a
   network in an attempt to remove an overlay topology, or create an
   unauthorized overlay.

o supporting-network: A malicious client could attempt to disrupt
   the logical structure of the model, resulting in lack of overall
   data integrity and making it more difficult to, for example,
   troubleshoot problems rooted in the layering of network
   topologies.

o node: A malicious client could attempt to remove or add a node
   from network, for example in order to sabotage the topology of a
   network overlay.

o supporting-node: A malicious client could attempt to change the
   supporting-node in order to sabotage the layering of an overlay.

These are the subtrees and data nodes and their sensitivity/
vulnerability in the ietf-network-topology module:

o link: A malicious client could attempt to remove a link from a
   topology, or add a new link, or manipulate the way the link is
   layered over supporting links, or modify the source or destination
   of the link. Either way, the structure of the topology would be
   sabotaged, which could, for example, result in an overlay topology
   that is less than optimal.

o termination-point: A malicious client could attempt to remove
   termination points from a node, or add "phantom" termination
   points to a node, or change the layering dependencies of
   termination points, again in an attempt to sabotage the integrity
   of a topology and potentially disrupt orderly operations of an
   overlay.

9. Contributors

   The data model presented in this paper was contributed to by more
   people than can be listed on the author list. Additional
   contributors include:

   o Vishnu Pavan Beeram, Juniper

   o Ken Gray, Cisco

   o Tom Nadeau, Brocade

   o Tony Tkacik

   o Kent Watsen, Juniper

   o Aleksandr Zhdankin, Cisco

10. Acknowledgements

   We wish to acknowledge the helpful contributions, comments, and
   suggestions that were received from Alia Atlas, Andy Bierman, Martin
   Bjorklund, Igor Bryskin, Benoit Claise, Susan Hares, Ladislav Lhotka,
   Carlos Pignataro, Juergen Schoenwaelder, Robert Wilton, Qin Wu, and
   Xian Zhang.

11. References

11.1. Normative References

[I-D.draft-ietf-netmod-revised-datastores]
          Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
          and R. Wilton, "A Revised Conceptual Model for YANG
          Datastores", I-D draft-ietf-netmod-revised-datastores-07,
          November 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to indicate
          requirement levels", RFC 2119, March 1997.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
          (TLS) Protocol Version 1.2", RFC 5246, August 2008.

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

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A.
          Bierman, "Network Configuration Protocol (NETCONF)",
          RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure
          Shell (SSH)", RFC 6242, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration
          Protocol (NETCONF) Access Control Model", RFC 6536, March
          2012.

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

[RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language",
          RFC 7950, August 2016.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF
          Protocol", RFC 8040, January 2017.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
          2119 Key Words", RFC 8174, May 2017.

11.2. Informative References

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

   [I-D.draft-ietf-i2rs-yang-l3-topology]
              Clemm, A., Medved, J., Varga, R., Liu, X.,
              Ananthakrishnan, H., and N. Bahadur, "A YANG Data Model
              for Layer 3 Topologies", I-D draft-ietf-i2rs-yang-l3-
              topology-16, December 2017.

   [I-D.draft-ietf-netconf-yang-push]
              Clemm, A., Voit, E., Gonzalez Prieto, A., Tripathy, A.,
              Nilsen-Nygaard, E., Bierman, A., and B. Lengyel,
              "Subscribing to YANG datastore push updates", I-D draft-
              ietf-netconf-yang-push-11, October 2017.

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

   [RFC1195] Callon, R., "Use of OSI IS-IS for Routing in TCP/IP and
              Dual Environments", RFC 1195, December 1990.

   [RFC2328] Moy, J., "OSPF Version 2", RFC 2328, April 1998.

   [RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V.,
              and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP
              Tunnels", RFC 3209, December 2001.

   [RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between
              Information Models and Data Models", RFC 3444, January
              2003.

   [RFC7223] Bjorklund, M., "A YANG Data Model for Interface
              Management", RFC 7223, May 2014.

   [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG",
              RFC 7951, August 2016.

   [RFC7952] Lhotka, L., "Defining and Using Metadata with YANG",
              RFC 7952, August 2016.

   [RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing
              Management", RFC 8022, November 2016.

   [RFC8242] Haas, J. and S. Hares, "I2RS Ephemeral State
              Requirements", RFC 8242, September 2017.

Appendix A.   Model Use Cases

A.1.   Fetching Topology from a Network Element

   In its simplest form, topology is learned by a network element (e.g.,
   a router) through its participation in peering protocols (IS-IS, BGP,
   etc.).   This learned topology can then be exported (e.g., to a
   Network Management System) for external utilization.   Typically, any
   network element in a domain can be queried for its topology and
   expected to return the same result.

   In a slightly more complex form, the network element may be a
   controller, either by nature of it having satellite or subtended
   devices hanging off of it, or in the more classical sense, such as
   special device designated to orchestrate the activities of a number
   of other devices (e.g., an optical controller).   In this case, the
   controller device is logically a singleton and must be queried
   distinctly.

   It is worth noting that controllers can be built on top of
   controllers to establish a topology incorporating of all the domains
   within an entire network.

   In all of the cases above, the topology learned by the network
   element is considered to be operational state data.   That is, the
   data is accumulated purely by the network element's interactions with
   other systems and is subject to change dynamically without input or
   consent.

A.2.   Modifying TE Topology Imported from an Optical Controller

   Consider a scenario where an Optical Transport Controller presents
   its topology in abstract TE Terms to a Client Packet Controller.
   This Customized Topology (that gets merged into the Client's native
   topology) contains sufficient information for the path computing
   client to select paths across the optical domain according to its
   policies.   If the Client determines (at any given point in time) that
   this imported topology does not exactly cater to its requirements, it
   may decide to request modifications to the topology.   Such
   customization requests may include addition or deletion of
   topological elements or modification of attributes associated with
   existing topological elements.   From the perspective of the Optical
   Controller, these requests translate into configuration changes to
   the exported abstract topology.

A.3. Annotating Topology for Local Computation

   In certain scenarios, the topology learned by a controller needs to
   be augmented with additional attributes before running a computation
   algorithm on it. Consider the case where a path-computation
   application on the controller needs to take the geographic
   coordinates of the nodes into account while computing paths on the
   learned topology. If the learned topology does not contain these
   coordinates, then these additional attributes must be configured on
   the corresponding topological elements.

A.4. SDN Controller-Based Configuration of Overlays on Top of Underlays

   In this scenario, an SDN controller (for example, Open Daylight)
   maintains a view of the topology of the network that it controls
   based on information that it discovers from the network. In
   addition, it provides an application in which it configures and
   maintains an overlay topology.

   The SDN Controller thus maintains two roles:

   o It is a client to the network.

   o It is a server to its own northbound applications and clients,
      e.g. an OSS.

   In other words, one system's client (or controller, in this case) may
   be another system's server (or managed system).

   In this scenario, the SDN controller maintains a consolidated data
   model of multiple layers of topology. This includes the lower layers
   of the network topology, built from information that is discovered
   from the network. It also includes upper layers of topology overlay,
   configurable by the controller's client, i.e. the OSS. To the OSS,
   the lower topology layers constitute "read-only" information. The
   upper topology layers need to be read-writable.

Appendix B. Companion YANG models for non-NMDA compliant
             implementations

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

   mirror modules ietf-network and ietf-network-topology defined earlier
   in this document. However, all data nodes are non-configurable.
   They represent state that comes into being by either learning
   topology information from the network, or by applying configuration
   from the mirrored modules.

   The companion modules, ietf-network-state and ietf-network-topology-
   state, are redundant and SHOULD NOT be supported by implementations
   that support NMDA. It is for this reason that the definitions are
   defined in an appendix.

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

B.1. YANG Model for Network State

   NOTE TO RFC EDITOR: Please change the date in the file name after the
   CODE BEGINS statement to the date of the publication when published.

```
<CODE BEGINS> file "ietf-network-state@2017-12-18.yang"
module ietf-network-state {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-state";
  prefix nw-s;

  import ietf-network {
    prefix nw;
    reference
      "draft-ietf-i2rs-yang-network-topo-20
      NOTE TO RFC EDITOR: Please replace above reference to
      draft-ietf-i2rs-yang-network-topo-20 with RFC
      number when published (i.e. RFC xxxx).";
  }

  organization
    "IETF I2RS (Interface to the Routing System) Working Group";

  contact
    "WG Web:     <http://tools.ietf.org/wg/i2rs/>
     WG List:    <mailto:i2rs@ietf.org>

     Editor:     Alexander Clemm
                 <mailto:ludwig@clemm.org>

     Editor:     Jan Medved
                 <mailto:jmedved@cisco.com>

     Editor:     Robert Varga
```

                 <mailto:robert.varga@pantheon.tech>

      Editor: Nitin Bahadur
                   <mailto:nitin_bahadur@yahoo.com>

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

      Editor: Xufeng Liu
                   <mailto:Xufeng_Liu@jabil.com>";

  description
    "This module defines a common base data model for a collection
     of nodes in a network. Node definitions are further used
     in network topologies and inventories. It represents
     information that is either learned and automatically populated,
     or information that results from applying configured netwok
     information configured per the ietf-network data model,
     mirroring the corresponding data nodes in this data model.

     The data model mirrors ietf-network, but contains only
     read-only state data. The data model is not needed when the
     underlying implementation infrastructure supports the Network
     Management Datastore Architecture (NMDA).

     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
     draft-ietf-i2rs-yang-network-topo-20;
     see the RFC itself for full legal notices.

     NOTE TO RFC EDITOR: Please replace above reference to
     draft-ietf-i2rs-yang-network-topo-20 with RFC
     number when published (i.e. RFC xxxx).";

  revision 2017-12-18 {
    description
      "Initial revision.
       NOTE TO RFC EDITOR:
       (1) Please replace the following reference

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

  grouping network-ref {
    description
      "Contains the information necessary to reference a network,
       for example an underlay network.";
    leaf network-ref {
      type leafref {
        path "/nw-s:networks/nw-s:network/nw-s:network-id";
      require-instance false;
      }
      description
        "Used to reference a network, for example an underlay
         network.";
    }
  }

  grouping node-ref {
    description
      "Contains the information necessary to reference a node.";
    leaf node-ref {
      type leafref {
        path "/nw-s:networks/nw-s:network[nw-s:network-id=current()"+
          "/../network-ref]/nw-s:node/nw-s:node-id";
        require-instance false;
      }
      description
        "Used to reference a node.
         Nodes are identified relative to the network they are
         contained in.";
    }
    uses network-ref;
  }

  container networks {
    config false;
    description
      "Serves as top-level container for a list of networks.";
    list network {
      key "network-id";
      description
        "Describes a network.
```

```
          A network typically contains an inventory of nodes,
          topological information (augmented through
          network-topology data model), as well as layering
          information.";
      container network-types {
        description
          "Serves as an augmentation target.
           The network type is indicated through corresponding
           presence containers augmented into this container.";
      }
      leaf network-id {
        type nw:network-id;
        description
          "Identifies a network.";
      }
      list supporting-network {
        key "network-ref";
        description
          "An underlay network, used to represent layered network
           topologies.";
        leaf network-ref {
          type leafref {
            path "/nw-s:networks/nw-s:network/nw-s:network-id";
            require-instance false;
          }
          description
            "References the underlay network.";
        }
      }
      list node {
        key "node-id";
        description
          "The inventory of nodes of this network.";
        leaf node-id {
          type nw:node-id;
          description
            "Identifies a node uniquely within the containing
             network.";
        }
        list supporting-node {
          key "network-ref node-ref";
          description
            "Represents another node, in an underlay network, that
             this node is supported by.  Used to represent layering
             structure.";
          leaf network-ref {
            type leafref {
              path "../../../nw-s:supporting-network/nw-s:network-ref";
```

```
                require-instance false;
                }
              description
                "References the underlay network that the
                 underlay node is part of.";
            }
            leaf node-ref {
              type leafref {
                path "/nw-s:networks/nw-s:network/nw-s:node/nw-s:node-id";
              require-instance false;
              }
              description
                "References the underlay node itself.";
            }
          }
        }
      }
    }
  }
}
```
<CODE ENDS>

B.2. YANG Data Model for Network Topology State

   NOTE TO RFC EDITOR: Please change the date in the file name after the
   CODE BEGINS statement to the date of the publication when published.

  <CODE BEGINS> file "ietf-network-topology-state@2017-12-18.yang"
  module ietf-network-topology-state {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-network-topology-state";
    prefix nt-s;

    import ietf-network-state {
      prefix nw-s;
      reference
        "draft-ietf-i2rs-yang-network-topo-20
        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-network-topo-20 with RFC
        number when published (i.e. RFC xxxx).";
    }
    import ietf-network-topology {
      prefix nt;
      reference
        "draft-ietf-i2rs-yang-network-topo-20
        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-network-topo-20 with RFC
        number when published (i.e. RFC xxxx).";
    }
```

      organization
        "IETF I2RS (Interface to the Routing System) Working Group";

      contact
        "WG Web: <http://tools.ietf.org/wg/i2rs/>
        WG List: <mailto:i2rs@ietf.org>

        Editor: Alexander Clemm
                    <mailto:ludwig@clemm.org>

        Editor: Jan Medved
                    <mailto:jmedved@cisco.com>

        Editor: Robert Varga
                    <mailto:robert.varga@pantheon.tech>

        Editor: Nitin Bahadur
                    <mailto:nitin_bahadur@yahoo.com>

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

        Editor: Xufeng Liu
                    <mailto:Xufeng_Liu@jabil.com>";

    description
       "This module defines a common base data model for network
        topology state, representing topology that is either learned,
        or topology that results from applying topology that has been
        configured per the ietf-network-topology data model, mirroring
        the corresponding data nodes in this data model. It augments
        the base network state data model with links to connect nodes,
        as well as termination points to terminate links on nodes.

        The data model mirrors ietf-network-topology, but contains only
        read-only state data. The data model is not needed when the
        underlying implementation infrastructure supports the Network
        Management Datastore Architecture (NMDA).

        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
     draft-ietf-i2rs-yang-network-topo-20;
     see the RFC itself for full legal notices.

     NOTE TO RFC EDITOR: Please replace above reference to
     draft-ietf-i2rs-yang-network-topo-20 with RFC
     number when published (i.e. RFC xxxx).";

   revision 2017-12-18 {
     description
       "Initial revision.
        NOTE TO RFC EDITOR:
        (1) Please replace the following reference
        to draft-ietf-i2rs-yang-network-topo-20 with
        RFC number when published (i.e. RFC xxxx).
        (2) Please replace the date in the revision statement with the
        date of publication when published.";
     reference
       "draft-ietf-i2rs-yang-network-topo-20";
   }

   grouping link-ref {
     description
       "References a link in a specific network.  While this grouping
        is not used in this module, it is defined here for the
        convenience of augmenting modules.";
     leaf link-ref {
       type leafref {
         path "/nw-s:networks/nw-s:network[nw-s:network-id=current()"+
           "/../network-ref]/nt-s:link/nt-s:link-id";
         require-instance false;
       }
       description
         "A type for an absolute reference a link instance.
          (This type should not be used for relative references.
          In such a case, a relative path should be used instead.)";
     }
     uses nw-s:network-ref;
   }

   grouping tp-ref {
     description
       "References a termination point in a specific node.  While
        this grouping is not used in this module, it is defined here
        for the convenience of augmenting modules.";
     leaf tp-ref {
       type leafref {
         path "/nw-s:networks/nw-s:network[nw-s:network-id=current()"+
```

```
            "/../network-ref]/nw-s:node[nw-s:node-id=current()/../"+
            "node-ref]/nt-s:termination-point/nt-s:tp-id";
          require-instance false;
        }
        description
          "A type for an absolute reference to a termination point.
           (This type should not be used for relative references.
           In such a case, a relative path should be used instead.)";
      }
      uses nw-s:node-ref;
    }

    augment "/nw-s:networks/nw-s:network" {
      description
        "Add links to the network data model.";
      list link {
        key "link-id";
        description
          "A network link connects a local (source) node and
           a remote (destination) node via a set of
           the respective node's termination points.
           It is possible to have several links between the same
           source and destination nodes.  Likewise, a link could
           potentially be re-homed between termination points.
           Therefore, in order to ensure that we would always know
           to distinguish between links, every link is identified by
           a dedicated link identifier.  Note that a link models a
           point-to-point link, not a multipoint link.";
        container source {
          description
            "This container holds the logical source of a particular
             link.";
          leaf source-node {
            type leafref {
              path "../../../nw-s:node/nw-s:node-id";
              require-instance false;
            }
            description
              "Source node identifier, must be in same topology.";
          }
          leaf source-tp {
            type leafref {
              path "../../../nw-s:node[nw-s:node-id=current()/../"+
                "source-node]/termination-point/tp-id";
              require-instance false;
            }
            description
              "Termination point within source node that terminates
```

```
              the link.";
          }
        }
        container destination {
          description
            "This container holds the logical destination of a
             particular link.";
          leaf dest-node {
            type leafref {
              path "../../../nw-s:node/nw-s:node-id";
            require-instance false;
            }
            description
              "Destination node identifier, must be in the same
               network.";
          }
          leaf dest-tp {
            type leafref {
              path "../../../nw-s:node[nw-s:node-id=current()/../"+
                "dest-node]/termination-point/tp-id";
              require-instance false;
            }
            description
              "Termination point within destination node that
               terminates the link.";
          }
        }
        leaf link-id {
          type nt:link-id;
          description
            "The identifier of a link in the topology.
             A link is specific to a topology to which it belongs.";
        }
        list supporting-link {
          key "network-ref link-ref";
          description
            "Identifies the link, or links, that this link
             is dependent on.";
          leaf network-ref {
            type leafref {
              path "../../../nw-s:supporting-network/nw-s:network-ref";
            require-instance false;
            }
            description
              "This leaf identifies in which underlay topology
               the supporting link is present.";
          }
          leaf link-ref {
```

```
            type leafref {
              path "/nw-s:networks/nw-s:network[nw-s:network-id="+
                "current()/../network-ref]/link/link-id";
              require-instance false;
            }
            description
              "This leaf identifies a link which is a part
               of this link's underlay. Reference loops in which
               a link identifies itself as its underlay, either
               directly or transitively, are not allowed.";
          }
        }
      }
    }
    augment "/nw-s:networks/nw-s:network/nw-s:node" {
      description
        "Augment termination points which terminate links.
         Termination points can ultimately be mapped to interfaces.";
      list termination-point {
        key "tp-id";
        description
          "A termination point can terminate a link.
           Depending on the type of topology, a termination point
           could, for example, refer to a port or an interface.";
        leaf tp-id {
          type nt:tp-id;
          description
            "Termination point identifier.";
        }
        list supporting-termination-point {
          key "network-ref node-ref tp-ref";
          description
            "This list identifies any termination points that
             the termination point is dependent on, or maps onto.
             Those termination points will themselves be contained
             in a supporting node.
             This dependency information can be inferred from
             the dependencies between links.  For this reason,
             this item is not separately configurable.  Hence no
             corresponding constraint needs to be articulated.
             The corresponding information is simply provided by the
             implementing system.";
          leaf network-ref {
            type leafref {
              path "../../../nw-s:supporting-node/nw-s:network-ref";
            require-instance false;
            }
            description
```

```
                    "This leaf identifies in which topology the
                     supporting termination point is present.";
              }
            leaf node-ref {
              type leafref {
                path "../../../nw-s:supporting-node/nw-s:node-ref";
              require-instance false;
              }
              description
                "This leaf identifies in which node the supporting
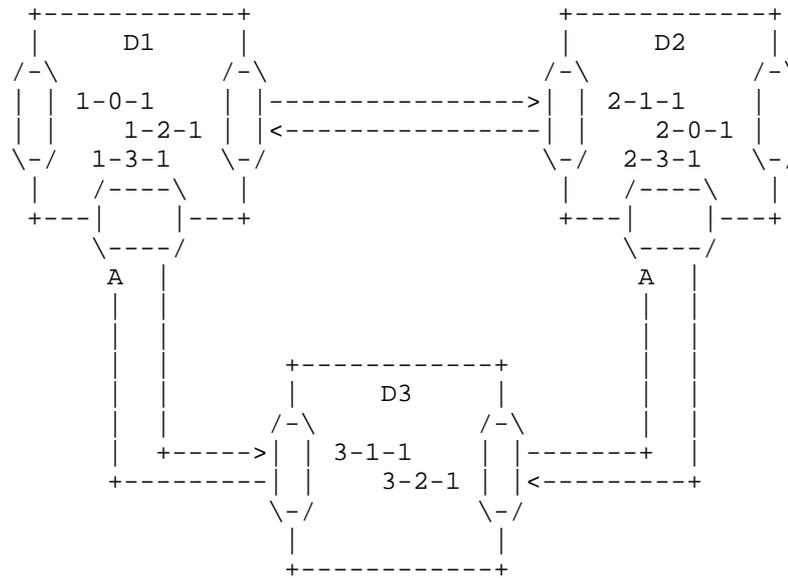                 termination point is present.";
            }
            leaf tp-ref {
              type leafref {
                path "/nw-s:networks/nw-s:network[nw-s:network-id="+
                   "current()/../network-ref]/nw-s:node[nw-s:node-id="+
                   "current()/../node-ref]/termination-point/tp-id";
                require-instance false;
              }
              description
                "Reference to the underlay node, must be in a
                 different topology";
            }
          }
        }
      }
    }
  }

  <CODE ENDS>
```

Appendix C. An Example

   This section contains an example of an instance data tree in JSON
   encoding [RFC7951]. The example instantiates ietf-network-topology
   (and ietf-network, which ietf-network-topology augments) for the
   topology that is depicted in the following diagram. There are three
   nodes, D1, D2, and D3. D1 has three termination points, 1-0-1,
   1-2-1, and 1-3-1. D2 has three termination points as well, 2-1-1,
   2-0-1, and 2-3-1. D3 has two termination points, 3-1-1 and 3-2-1.
   In addition there are six links, two between each pair of nodes with
   one going in each direction.

```
              +-----------+                    +-----------+
              |    D1     |                    |    D2     |
             /-\         /-\                  /-\         /-\
             | | 1-0-1   | |---------------->| | 2-1-1   | |
             | |   1-2-1 | |<----------------| |   2-0-1 | |
             \-/  1-3-1  \-/                  \-/  2-3-1  \-/
              |  /----\   |                    |  /----\   |
             +---|    |---+                   +---|    |---+
                 \----/                           \----/
                 A  |                             A  |
                 |  |                             |  |
                 |  |                             |  |
                 |  |     +-----------+           |  |
                 |  |     |    D3     |           |  |
                 |  |    /-\         /-\          |  |
                 |  +----->| | 3-1-1   | |-------+  |
                 +---------| |   3-2-1 | |<---------+
                          \-/         \-/
                           |           |
                          +-----------+
```

                  Figure 7: A network topology example

   The corresponding instance data tree is depicted below:

```
   {
     "ietf-network:networks": {
       "network": [
         {
           "network-types": {
           },
           "network-id": "otn-hc",
           "node": [
             {
               "node-id": "D1",
               "termination-point": [
                 {
                   "tp-id": "1-0-1"
                 },
                 {
                   "tp-id": "1-2-1"
                 },
                 {
                   "tp-id": "1-3-1"
                 }
               ]
             },
```

```
                {
                  "node-id": "D2",
                  "termination-point": [
                    {
                      "tp-id": "2-0-1"
                    },
                    {
                      "tp-id": "2-1-1"
                    },
                    {
                      "tp-id": "2-3-1"
                    }
                  ]
                },
                {
                  "node-id": "D3",
                  "termination-point": [
                    {
                      "tp-id": "3-1-1"
                    },
                    {
                      "tp-id": "3-2-1"
                    }
                  ]
                }
              ],
              "ietf-network-topology:link": [
                {
                  "link-id": "D1,1-2-1,D2,2-1-1",
                  "destination": {
                    "source-node": "D1",
                    "source-tp": "1-2-1"
                  }
                  "destination": {
                    "dest-node": "D2",
                    "dest-tp": "2-1-1"
                  }
                },
                {
                  "link-id": "D2,2-1-1,D1,1-2-1",
                  "destination": {
                    "source-node": "D2",
                    "source-tp": "2-1-1"
                  }
                  "destination": {
                    "dest-node": "D1",
                    "dest-tp": "1-2-1"
                  }
```

```
            },
            {
              "link-id": "D1,1-3-1,D3,3-1-1",
              "destination": {
                "source-node": "D1",
                "source-tp": "1-3-1"
              }
              "destination": {
                "dest-node": "D3",
                "dest-tp": "3-1-1"
              }
            },
            {
              "link-id": "D3,3-1-1,D1,1-3-1",
              "destination": {
                "source-node": "D3",
                "source-tp": "3-1-1"
              }
              "destination": {
                "dest-node": "D1",
                "dest-tp": "1-3-1"
              }
            },
            {
              "link-id": "D2,2-3-1,D3,3-2-1",
              "destination": {
                "source-node": "D2",
                "source-tp": "2-3-1"
              }
              "destination": {
                "dest-node": "D3",
                "dest-tp": "3-2-1"
              }
            },
            {
              "link-id": "D3,3-2-1,D2,2-3-1",
              "destination": {
                "source-node": "D3",
                "source-tp": "3-2-1"
              }
              "destination": {
                "dest-node": "D2",
                "dest-tp": "2-3-1"
              }
            }
          ]
        }
      ]
```

```
      }
    }
```

                        Figure 8: Instance data tree

Authors' Addresses

   Alexander Clemm
   Huawei

   EMail: ludwig@clemm.org

   Jan Medved
   Cisco

   EMail: jmedved@cisco.com

   Robert Varga
   Pantheon Technologies SRO

   EMail: robert.varga@pantheon.tech

   Nitin Bahadur
   Bracket Computing

   EMail: nitin_bahadur@yahoo.com

   Hariharan Ananthakrishnan
   Packet Design

   EMail: hari@packetdesign.com

   Xufeng Liu
   Jabil

   EMail: Xufeng_Liu@jabil.com

i2rs   M. Wang, Ed.
Internet-Draft   Huawei
Intended status: Informational   R. Gu
Expires: May 2, 2018   China Mobile
                                                          Victor. Lopez
                                                             Telefonica
                                                                  S. Hu
                                                          China Mobile
                                                       October 29, 2017

      Information Model of Control-Plane and User-Plane separation BNG
             draft-wcg-i2rs-cu-separation-infor-model-02

Abstract

   To improve network resource utilization and reduce the operation
   expense, the Control-Plane and User-Plane separation conception is
   raised [I-D.gu-nfvrg-cloud-bng-architecture]. This document
   describes the information model for the interface between Control-
   Plane and User-Plane separation BNG. This information model may
   involve both control channel interface and configuration channel
   interface. The interface for control channel allows the Control-
   Plane to send several flow tables to the User-Plane, such as user's
   information table, user's interface table, and user's QoS table, etc.
   And it also allows the User-Plane to report the resources and
   statistics information to the Control-Plane. The interface for
   configuration channel is in charge of the version negotiation of
   protocols between the Control-Plane and User-Plane, the configuration
   for devices of Control-Plane and User-Plane, and the reports of User-
   Plane's capabilities, etc. The information model defined in this
   document enables defining a standardized data model. Such a data
   model can be used to define an interface to the CU separation BNG.

Status of This Memo

This Internet-Draft will expire on May 2, 2018.

Copyright Notice

Table of Contents

1. Introduction

   The rapid development of new services, such as 4K, IoT, etc, and
   increasing numbers of home broadband service users present some new
   challenges for BNGs such as:

      Low resource utilization: The traditional BNG acts as both a
      gateway for user access authentication and accounting and an IP
      network's Layer 3 edge. The mutually affecting nature of the
      tightly coupled control plane and forwarding plane makes it
      difficult to achieve the maximum performance of either plane.

      Complex management and maintenance: Due to the large numbers of
      traditional BNGs, a network must have each device configured one
      at a time when deploying global service policies. As the network
      expands and new services are introduced, this deployment mode will
      cease to be feasible as it is unable to manage services
      effectively and rectify faults rapidly.

      Slow service provisioning: The coupling of control plane and
      forwarding plane, in addition to a distributed network control
      mechanism, means that any new technology has to rely heavily on
      the existing network devices.

   To address these challenges, cloud-based BNG with CU separation
   conception is raised [I-D.gu-nfvrg-cloud-bng-architecture]. The main
   idea of Control-Plane and User-Plane separation method is to extract
   and centralize the user management functions of multiple BNG devices,
   forming an unified and centralized control plane (CP). And the
   traditional router's Control Plane and Forwarding Plane are both
   preserved on BNG devices in the form of a user plane (UP).

   This document describes an information model for the interface
   between Control-Plane and User-Plane separation BNG. This
   information model may involve both control channel interface and
   configuration channel interface. The interface for control channel
   allows the Control-Plane to send several flow tables to the User-
   Plane, such as user's information table, user's interface table, and
   user's QoS table, etc. And it also allows User-Plane to report the
   resources and statistics information to the Control-Plane. The
   interface for configuration channel is in charge of the version
   negotiation of protocols between the Control-Plane and User-Plane,
   the configuration for the devices of Control-Plane and User-Plane,
   and the report of User-Plane's capabilities, etc. The information
   model defined in this document enables defining a standardized data
   model. Such a data model can be used to define an interface to the
   CU separation BNG.

2. Concept and Terminology

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

2.1. Terminology

   BNG: Broadband Network Gateway. A broadband remote access server
   (BRAS, B-RAS or BBRAS) routes traffic to and from broadband remote
   access devices such as digital subscriber line access multiplexers
   (DSLAM) on an Internet service provider's (ISP) network. BRAS can
   also be referred to as a Broadband Network Gateway (BNG).

   CP: Control Plane. CP is a user control management component which
   supports the management of UP's resources such as the user entry and
   forwarding policy

   UP: User Plane. UP is a network edge and user policy implementation
   component. The traditional router's Control Plane and Forwarding
   Plane are both preserved on BNG devices in the form of a user plane.

3. Control Plane and User Plane separation BNG Information Model
    Overview

   Briefly, a CU separation BNG is made up of a centralized CP and a set
   of UPs. The CP is a user control management component which supports
   to manage UP's resources such as the user entry and forwarding
   policy, for example, the access bandwidth and priority management.
   And the UP is a network edge and user policy implementation
   component. It can support the forwarding plane functions on
   traditional BNG devices, such as traffic forwarding, QoS, and traffic
   statistics collection, and it can also support the control plane
   functions on traditional BNG devices, such as routing, multicast,
   etc.

   The following figure describes the architecture of CU separation BNG

```
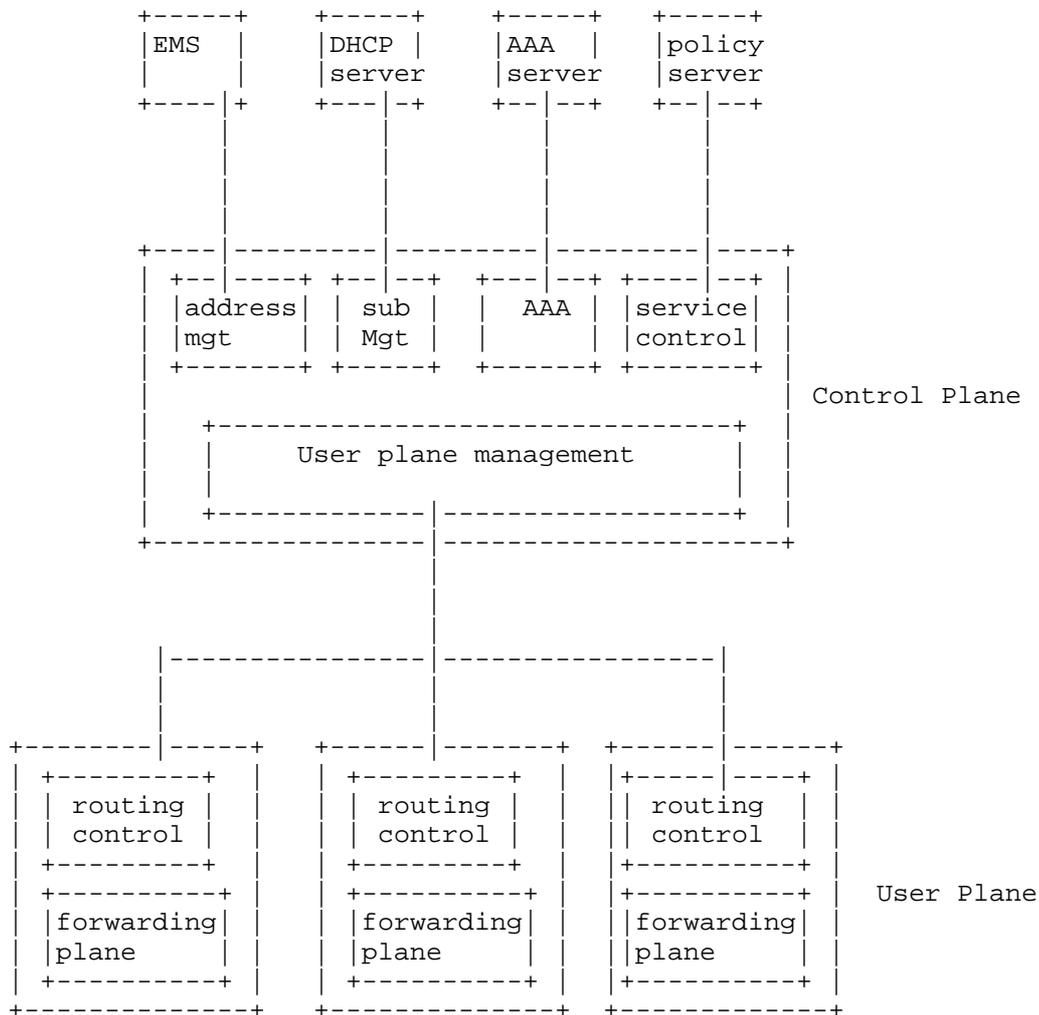        +-----+    +-----+    +-----+    +-----+
        |EMS  |    |DHCP |    |AAA  |    |policy
        |     |    |server    |server    |server
        +----|+    +---|-+    +--|--+    +--|--+
             |         |         |          |
             |         |         |          |
             |         |         |          |
             |         |         |          |
        +----|---------|---------|---------|----+
        | +--|----+ +--|--+ +---|--+ +----|--+ |
        | |address| | sub | | AAA  | |service| |
        | |mgt    | | Mgt | |      | |control| |
        | +-------+ +-----+ +------+ +-------+ |
        |                                      |  Control Plane
        |   +------------------------------+   |
        |   |    User plane management     |   |
        |   |                              |   |
        |   +-------------|----------------+   |
        +-----------------|--------------------+
                          |
                          |
                          |
            |-------------|----------------|
            |             |                |
            |             |                |
    +--------|-----+ +------|------+ +------|------+
    | +---------+ | | +---------+ | |+-----|----+ |
    | | routing | | | | routing | | || routing  | |
    | | control | | | | control | | || control  | |
    | +---------+ | | +---------+ | |+----------+ |
    | +----------+ | | +----------+ | |+----------+ |
    | |forwarding| | | |forwarding| | ||forwarding| |
    | |plane     | | | |plane     | | ||plane     | |
    | +----------+ | | +----------+ | |+----------+ |   User Plane
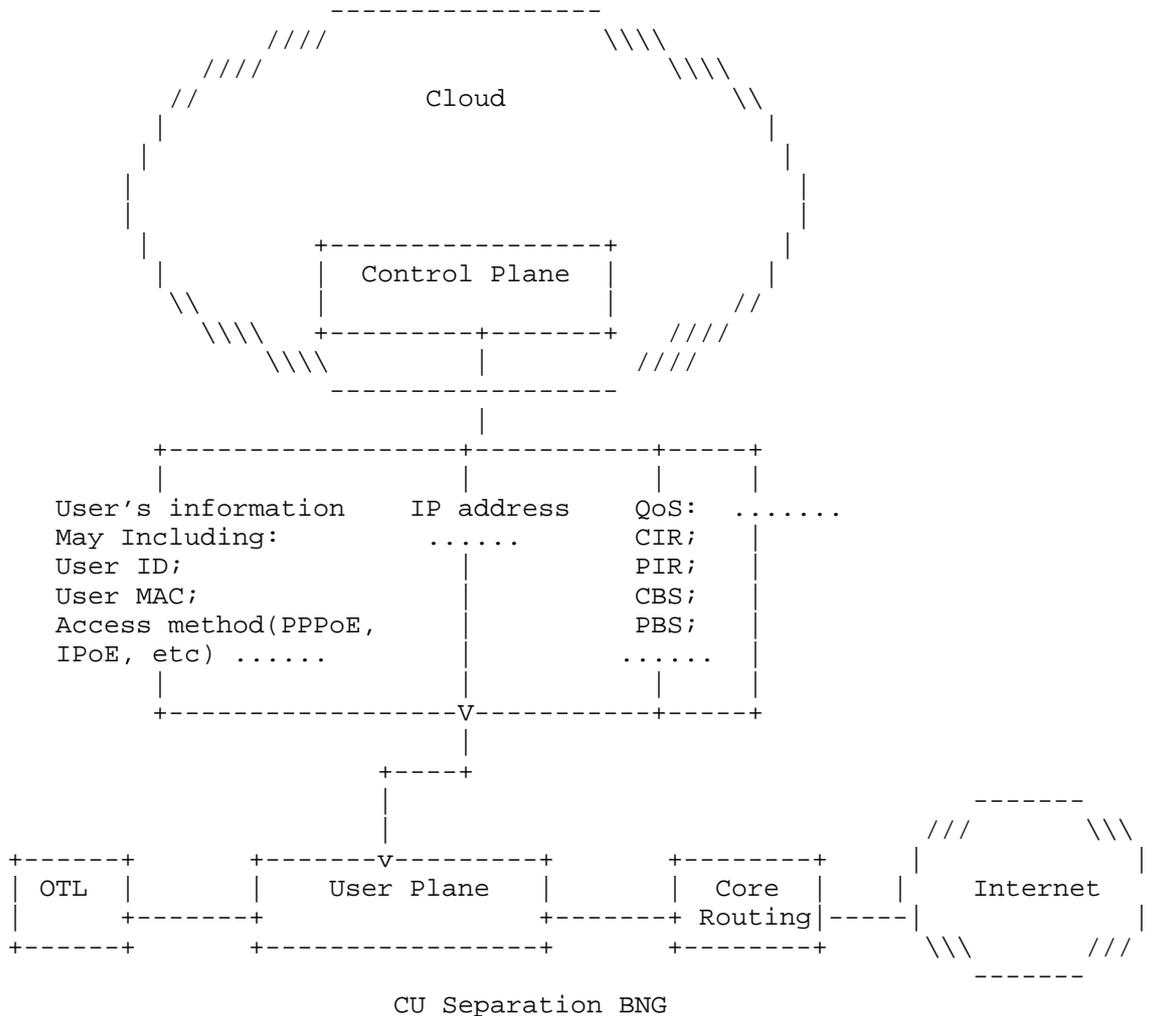    +--------------+ +--------------+ +-------------+
```

   The CU separated BNG is shown in above figure. The BNG Control Plane
   could be virtualized and centralized, which provides significant
   benefits such as centralized session management, flexible address
   allocation, high scalability for subscriber management capacity, and
   cost-efficient redundancy, etc. The functional components inside the
   BNG Service Control Plane can be implemented as VNFs and hosted in a
   NFVI.

   The User Plane Management module in the BNG control plane centrally
   manages the distributed BNG User Planes (e.g. load balancing), as
   well as the setup, deletion, maintenance of channels between Control

Planes and User Planes. Other modules in the BNG control plane, such
as address management, AAA, and etc., are responsible for the
connection with outside subsystems in order to fulfill the service.
The routing control and forwarding Plane in the BNG User Plane
(local) could be distributed across the infrastructure.

3.1. Service Data Model Usage

The idea of the information model is to propose a set of generic and
abstract information models. The models are intended to be used in
both Control Plane and User Planes. A typical scenario would be that
this model can be used as a compendium for the interface between
Control Plane and User Planes of CU separation BNG, that
corresponding data model or TLVs can be defined to realize the
communication between the Control Plane and User Planes.

```
                     -----------------
                 ////                   \\\\
               ////                       \\\\
             //              Cloud          \\ |
            |                                  |
            |                                  |
           |                                    |
           |                                    |
           |        +-----------------+         |
           |        | Control Plane   |         |
            \\      |                 |      //
             \\\\   +---------+-------+   ////
               \\\\           |         ////
                     ----------------
                                |
         +-----------------+----------+-----+
         |                 |          |     |
        User's information   IP address   QoS:  .......
        May Including:       ......       CIR;  |
        User ID;             |            PIR;  |
        User MAC;            |            CBS;  |
        Access method(PPPoE, |            PBS;  |
        IPoE, etc) ......    |            ......|
         |                 |          |     |
         +-----------------V----------+-----+
                           |
                   +----+
                   |                              -------
                   |                           ///         \\\
+------+     +-------V---------+     +--------+   |           |
| OTL  |     |  User Plane    |     | Core   |   |  Internet  |
|      +-------+              +-------+ Routing|-----|          |
+------+     +----------------+     +--------+   \\\         ///
                                                    -------
                    CU Separation BNG
```

As shown in above figure, when users access to the BNG network, the
control plane solicits these users' information (such as user's ID,
user's MAC, user's access methods, for example via PPPoE/IPoE),
associates them with available bandwidth which are reported by User
planes, and based on the service's requirement to generate a set of
tables, which may include user's information, user's IP address, and
QoS, etc. Then the control plane can transmit these tables to the
User planes. User planes receive these tables, parses it, matches
these rules, and then performs corresponding actions.

4. Information Model

   This section specifies the information model in Routing Backus-Naur
   Form [I-D.gu-nfvrg-cloud-bng-architecture]. This grammar intends to
   help readers better understand the English text description in order
   to derive a data model. However it may not provide all the details
   provided by the English text. When there is a lack of clarity in
   grammar the English text will take precedence.

   This section describes information model that represents the concept
   of the interface of CU separation BNG which is languages and
   protocols neutral.

   The following figure describes the Overview of Information Model for
   CU separation BNG.

 <cu-separation-bng-infor-model>::=<control-plane-information-model>
                                   <user-plane-information-model>

 <control-plane-information-model>::=<user-related-infor-model>
                                     <interface-related-infor-model>
                                     <device-related-infor-model>

 <user-related-infor-model>::= <user-basic-information>
                               [<ipv4-informatiom>]|[<ipv6-information>]
                               [<qos-information>]

 <user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                               [<ACCESS_TYPE>][<SESSION_ID>]
                               [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                               <USER_INTERFACE>

 <ipv4-informatiom>::=<USER_ID><USER_IPV4>
                      <MASK_LENGTH><GATEWAY>
                      <VRF>

 <ipv6-information>::=<USER_ID>(<USER_IPV6>
                      <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                      <VRF>

 <qos-information>::=<USER_ID>
                     (<CIR><PIR><CBS><PBS>)
                     [<QOS_PROFILE>]

 <interface-related-infor-model>::=<interface-information>

 <interface-information>::=<IFINDEX><BAS_ENABLE>
                           <service-type>

```
  <service-type>::=<PPP_Only><IPV4_TRIG>
                   <IPV6_TRIG><ND-TRIG>
                   <ARP_PROXY>

  <device-related-infor-model>::=<address-field-distribute>

  <address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                               <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                               <IF_INDEX><MASK_LENGTH>

  <user-plane-information-model>::=<port-resources-infor-model>
                                   <traffic-statistics>

  <port-resource-information>::=<IF_INDEX><IF_NAME>
                               <IF_TYPE><LINK_TYPE>
                               <MAC_ADDRESS><IF_PHY_STATE>
                               <MTU>
  <traffic-statistics-information>::=<USER_ID><STATISTICS_TYPE>
                                     <INGRESS_STATIISTICS_PACKETS>
                                     <INGRESS STATISTICS_BYTES>
                                     <EGRESS_STATISTICS_PACKETS>
                                     <EGRESS_STATISTICS_BYTES>
```

4.1. Information Model for Control-Plane

   This section describes information model for the Control-Plane (CP).
   As mentioned in section 3, the Control Plane is a user control
   management component which manages the user's information, User-
   Plane's resources and forwarding policy, etc. The control plane can
   generate several tables which contain a set of rules based on the
   resources and specific requirements of user's service. After that,
   the control plane sends the tables to User Planes, and User planes
   receive the tables, parse them, match the rules, and then perform
   corresponding actions.

   The Routing Backus-Naur Form grammar below illustrates the
   Information model for Control-Plane:

```
<control-plane-information-model>::=<user-related-infor-model>
                                    <interface-related-infor-model>
                                    <device-related-infor-model>

<user-related-infor-model>::= <user-basic-information>
                              [<ipv4-informatiom>]|[<ipv6-information>]
                              [<qos-information>]

<user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                              [<ACCESS_TYPE>][<SESSION_ID>]
                              [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                              <USER_INTERFACE>

<ipv4-informatiom>::=<USER_ID><USER_IPV4>
                     <MASK_LENGTH><GATEWAY>
                     <VRF>

<ipv6-information>::=<USER_ID>(<USER_IPV6>
                     <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                     <VRF>

<qos-information>::=<USER_ID>
                    (<CIR><PIR><CBS><PBS>)
                    [<QOS_PROFILE>]

<interface-related-infor-model>::=<interface-information>

<interface-information>::=<IFINDEX><BAS_ENABLE>
                          <service-type>

<service-type>::=<PPP_Only><IPV4_TRIG>
                 <IPV6_TRIG><ND-TRIG>
                 <ARP_PROXY>

<device-related-infor-model>::=<address-field-distribute>

<address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                             <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                             <IF_INDEX><MASK_LENGTH>
```

   user-related-infor-model: present the attributes which can describe
   the user's profile, such as user's basic information, qos, and IP
   address, etc.

   interface-related-infor-model: present the attributes which relate to
   some physical/virtual interface. This model can be used to indicate
   which kinds of service can be supported by interfaces.

device-related-infor-model: present the attributes which relate to specific device. For example the control plane can manage and distribute the users, which belong to same subnet, to some specific devices. And the user plane's devices provide corresponding service for these users.

4.1.1. User-Related Information

   The user related information are a bunch of attributes which may bind to specific users. For example, the control plane can use a unified ID to distinguish different users and distribute the IP address and QoS rules to a specific user. In this section, the user related information models are presented. The user related information models include the user information model, IPv4/IPv6 information model, QoS information model, etc.

   The Routing Backus-Naur Form grammar below illustrates the user related information model:

```
<user-related-infor-model>::= <user-basic-information>
                              [<ipv4-informatiom>]|[<ipv6-information>]
                              [<qos-information>]

<user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                              [<ACCESS_TYPE>][<SESSION_ID>]
                              [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                              <USER_INTERFACE>

<ipv4-informatiom>::=<USER_ID><USER_IPV4>
                     <MASK_LENGTH><GATEWAY>
                     <VRF>

<ipv6-information>::=<USER_ID>(<USER_IPV6>
                     <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                     <VRF>

<qos-information>::=<USER_ID>
                    (<CIR><PIR><CBS><PBS>)
                    [<QOS_PROFILE>]
```

4.1.1.1. User Basic Information Model

   The User Basic Information model contains a set of attributes to describe the basic information of a specific user, such as user's mac address, access type (via PPPoE, IPoE, etc), inner vlan ID, outer vlan ID, etc.

The Routing Backus-Naur Form grammar below illustrates the user basic information model:

```
<user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                             [<ACCESS_TYPE>][<SESSION_ID>]
                             [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                             <USER_INTERFACE>
```

USER_ID: is the identifier of user. This parameter is a unique and mandatory, it can be used to distinguish different users.

MAC_ADDRESS: is the MAC address of the user.

ACCESS_TYPE: This attribute is an optional parameter. It can be used to indicate the protocol be used for user's accessing, such as PPPoE, IPoE, etc.

SESSION_ID: This attribute is an optional parameter. It can be used as the identifier of PPPoE session.

INNER_VLAN-ID: The identifier of user's inner VLAN.

OUTER_VLAN_ID: The identifier of user's outer VLAN.

USER_INTERFACE: This attribute specifies the binding interface of a specific user. The ifIndex of the interface MAY be included. This is the 32-bit ifIndex assigned to the interface by the device as specified by the Interfaces Group MIB [RFC2863]. The ifIndex can be utilized within a management domain to map to an actual interface, but it is also valuable in public applications [RFC5837]. The ifIndex can be used as an opaque token to discern which interface of User-Plane is providing corresponding service for specific user.

4.1.1.2. IPv4 Information Model

The IPv4 information model presents the user's IPv4 parameters. It is an optional constructs. The Routing Backus-Naur Form grammar below illustrates the user's IPv4 information model:

```
<ipv4-informatiom>::=<USER_ID><USER_IPV4>
                     <MASK_LENGTH><GATEWAY>
                     <VRF>
```

USER_ID: is the identifier of user. This parameter is unique and mandatory. This attribute is used to distinguish different users. And it collaborates with other IPv4 parameters to present the user's IPv4 information.

USER_IPV4: This attribute specifies the user's IPv4 address, and it's usually used in user plane discovery and ARP reply message.

MASK_LENGTH: This attribute specifies the user's subnet masks lengths which can identify a range of IP addresses that are on the same network.

GATEWAY: This attribute specifies the user's gateway, and it's usually used in User Plane discovery and ARP reply message.

VRF: is the identifier of VRF instance.

4.1.1.3. IPv6 Information Model

The IPv6 information model presents the user's IPv6 parameters. It is an optional constructs. The Routing Backus-Naur Form grammar below illustrates the user's IPv6 information model:

       <ipv6-information>::=<USER_ID>(<USER_IPV6>
                       <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                       <VRF>

USER_ID: is the identifier of user. This parameter is unique and mandatory. This attribute is used to distinguish different users. And it collaborates with other IPv6 parameters to present the user's IPv4 information.

USER_IPV6: This attribute specifies the user's IPv6 address, and it usually be used in neighbor discovery (ND discovery).

PREFIX_LEN: This attribute specifies the user's subnet prefix lengths which can identify a range of IP addresses that are on the same network.

PD_ADDRESS: In IPv6 networking, DHCPv6 prefix delegation is used to assign a network address prefix and automate configuration and provisioning of the public routable addresses for the network. This attribute specifies the user's DHCPv6 prefix delegation address, and it's usually used in neighbor discovery (ND discovery).

PD_PREFIX_LEN: This attribute specifies the user's DHCPv6 delegation prefix length, and it's usually used in neighbor discovery (ND discovery).

VRF: is the identifier of VRF instance

4.1.1.4. QoS Information Model

   In CU separation BNG, the Control-Plane (CP) generates the QoS table
   base on UP's bandwidth resources and specific QoS requirements of
   user's services. This table contains a set of QoS matching rules
   such as user's committed information rate, peak information rate,
   committed burst size, etc. And it is an optional constructs. The
   Routing Backus-Naur Form grammar below illustrates the user's qos
   information model:

   <qos-information>::=<USER_ID>
                    (<CIR><PIR><CBS><PBS>)
                    [<QOS_PROFILE>]

   USER_ID: is the identifier of user. This parameter is unique and
   mandatory. This attribute is used to distinguish different users.
   And it collaborates with other qos parameters to present the user's
   qos information.

   CIR: In BNG network, the Committed Information Rate (CIR) is the
   bandwidth for a user guaranteed by an internet service provider to
   work under normal conditions. This attribute is used to indicate the
   user's committed information rate, and it usually collaborates with
   other qos attributes (such as PIR, CBS, PBS, etc) to present the
   user's QoS profile.

   PIR: Peak Information Rate (PIR) is a burstable rate set on routers
   and/or switches that allows throughput overhead. This attribute is
   used to indicate the user's peak information rate, and it usually
   collaborate with other QoS attributes (such as CIR, CBS, PBS, etc) to
   present the user's QoS profile.

   CBS: The Committed Burst Size (CBS) specifies the relative amount of
   reserved buffers for a specific ingress network's forwarding class
   queue or egress network's forwarding class queue. This attribute is
   used to indicate the user's committed burst size, and it usually
   collaborates with other qos attributes (such as CIR, PIR, PBS, etc)
   to present the user's QoS profile.

   PBS: The Peak Burst Size (PBS) sepcifies the maximum size of the
   first token bucket. This attribute is used to indicate the user's
   peak burst size, and it usually collaborate with other qos attributes
   (such as CIR, PIR, CBS, etc) to present the user's QoS profile.

   QOS_PROFILE: This attribute specifies the standard profile provided
   by the operator. It can be used as a QoS template which is defined

as a list of classes of services and associated properties. The
properties may include:

    o Rate-limit: used to rate-limit the class of service. The value
    is expressed as a percentage of the global service bandwidth.

    o latency: used to define the latency constraint of the class.
    The latency constraint can be expressed as the lowest possible
    latency or a latency boundary expressed in milliseconds.

    o jitter: used to define the jitter constraint of the class. The
    jitter constraint can be expressed as the lowest possible jitter
    or a jitter boundary expressed in microseconds.

    o bandwidth: used to define a guaranteed amount of bandwidth for
    the class of service. It is expressed as a percentage.

4.1.2. Interface Related Information

This model contains the necessary information for the interface. It
is used to indicate which kind of service can be supported by this
interface. The Routing Backus-Naur Form grammar below illustrates
the interface related information model:

```
<interface-related-infor-model>::=<interface-information>

<interface-information>::=<IFINDEX><BAS_ENABLE>
                         <service-type>

<service-type>::=<PPP_Only><IPV4_TRIG>
                 <IPV6_TRIG><ND-TRIG>
                 <ARP_PROXY>
```

4.1.2.1. Interface Information Model

The interface model mentioned here is a logical construct that
identifies a specific process or a type of network service. In CU
separation BNG network, the Control-Plane (CP) generates the
Interface-Infor table based on the available resources, which are
received from the User-Plane (UP), and the specific requirements of
user's services.

The Routing Backus-Naur Form grammar below illustrates the interface
information model:

```
<interface-information>::=<IFINDEX><BAS_ENABLE>
                         <service-type>

<service-type>::=<PPP_Only><IPV4_TRIG>
                 <IPV6_TRIG><ND-TRIG>
                 <ARP_PROXY>
```

IFINDEX: The IfIndex is the 32-bit index assigned to the interface by
the device as specified by the Interfaces Group MIB [RFC2863]. The
ifIndex can be utilized within a management domain to map to an
actual interface, but it is also valuable in public applications.
The ifIndex can be used as an opaque token to discern which interface
of User-Plane is providing corresponding service for specific user.

BAS_ENABLE: This is a flag, and if it is TRUE, the BRAS is enabled on
this interface.

PPP_Only: This is a flag, and if it is TRUE, the interface only
supports PPP user.

IPV4_TRIG: This is a flag, and if it is TRUE, the interface supports
that the user can be triggered to connect the internet by using IPv4
message.

IPV6_TRIG: This is a flag, and if it is TRUE, the interface supports
that the user can be triggered to connect the internet by using IPv6
message.

ND-TRIG: This is a flag, and if it is TRUE, the interface supports
that the user can be triggered to connect the internet by using
neighbor discovery message.

ARP_PROXY: This is a flag, and if it is TRUE, the ARP PROXY is
enabled on this interface.

4.1.3. Device Related Information

The device related information model presents the attributes which
related to specific device. For example the control plane can manage
and distribute the users, who belong to same subnet, to some specific
devices. And then the user plane's devices can provide corresponding
service for these users. The Routing Backus-Naur Form grammar below
illustrates the device related information model:

```
<device-related-infor-model>::=<address-field-distribute>

<address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                             <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                             <IF_INDEX><MASK_LENGTH>
```

4.1.3.1. Address field distribute Table

   In CU separation BNG information model, the Control-Plane (CP)
   generates and sends this Address field distribute table to UP. Based
   on this table, the user-plane's devices can be divided into several
   blocks, and each block is in charge of working for users with the
   same subnet. The Routing Backus-Naur Form grammar below illustrates
   the address field distribute information model:

```
<address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                             <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                             <IF_INDEX><MASK_LENGTH>
```

4.2. Information Model for User Plane

   This section describes information model for the interface of User
   Plane (UP). As mentioned in section 3, the UP is a network edge and
   user policy implementation component. It supports: Forwarding plane
   functions on traditional BNG devices, including traffic forwarding,
   QoS, and traffic statistics collection and Control plane functions on
   traditional BNG devices, including routing, multicast, and MPLS.

   In CU separation BNG information model, the CP generates tables and
   provides the rules. The UP plays two roles:

   1. It receives these tables, parses it, and matches these rules,
       then performs corresponding actions.

   2. It also generates several tables to report the available
       resources (such as usable interfaces, etc) and statistical
       information to CP.

   The Routing Backus-Naur Form grammar below illustrates the User Plane
   information model:

```
<user-plane-information-model>::=<port-resources-infor-model>
                               <traffic-statistics>

port-resource-information>::=<IF_INDEX><IF_NAME>
                            <IF_TYPE><LINK_TYPE>
                            <MAC_ADDRESS><IF_PHY_STATE>
                            <MTU>
<traffic-statistics-information>::=<USER_ID><STATISTICS_TYPE>
                                  <INGRESS_STATIISTICS_PACKETS>
                                  <INGRESS STATISTICS_BYTES>
                                  <EGRESS_STATISTICS_PACKETS>
                                  <EGRESS_STATISTICS_BYTES>
```

4.2.1. Port Resources of UP

The User Plane can generate the network resource table, which
contains a bunch of attributes to present the available network
resources, for example the usable interfaces.

The Figure below illustrates the Port Resources Information Table of
User-Plane:

```
<port-resource-information>::<IF_INDEX><IF_NAME>
                            <IF_TYPE><LINK_TYPE>
                            <MAC_ADDRESS><IF_PHY_STATE>
                            <MTU>
```

IFINDEX: IfIndex is the 32-bit index assigned to the interface by the
device as specified by the Interfaces Group MIB [RFC2863]. The
ifIndex can be utilized within a management domain to map to an
actual interface, but it is also valuable in public applications.
The ifIndex can be used as an opaque token to discern which interface
of User-Plane is available.

IF_NAME: the textual name of the interface. The value of this object
should be the name of the interface as assigned by the local device
and should be suitable for use in commands entered at the device's
'console'. This might be a text name, such as 'le0' or a simple port
number, such as '1', depending on the interface naming syntax of the
device. If several entries in the ifTable together represent a
single interface as named by the device, then each will have the same
value of ifName.

IF_TYPE: the type of interface, such as Ethernet, GE, Eth-Trunk, etc.

LINK_TYPE: This attribute specifies the type of link, such as point-to-point, broadcast, multipoint, point-to-multipoint, private and public (accessibility and ownership), etc.

MAC_ADDRESS: This attribute specifies the available interface's MAC address.

IF_PHY_STATE: The current operational state of the interface. This is an enumeration type node:

    1- Up: ready to pass packets;

    2- Down

    3- Testing: in some test mode;

    4- Unknow: status cannot be determined for some reason;

    5- Dormant;

    6- Not present: some component is missing.

MTU: This attribute specifies the available interface's MTU (Maximum Transmission Unit).

4.2.2. Traffic Statistics Infor

The user-plane also generates the traffic statistics table to report the current traffic statistics.

The Figure below illustrates the Traffic Statistics Infor model of User-Plane:

```
<traffic-statistics-information>::=<USER_ID><STATISTICS_TYPE>
                                  <INGRESS_STATIISTICS_PACKETS>
                                  <INGRESS STATISTICS_BYTES>
                                  <EGRESS_STATISTICS_PACKETS>
                                  <EGRESS_STATISTICS_BYTES>
```

USER_ID: is the identifier of user. This parameter is unique and mandatory. This attribute is used to distinguish different users. And it collaborates with other statistics parameters such as ingress packets, egress packets, etc, to report the user's status profile.

STATISTICS_TYPE: This attributes specifies the traffic type such as IPv4, IPv6, etc.

INGRESS_STATIISTICS_PACKETS: This attribute specifies the Ingress
Statistics Packets of specific user.

INGRESS STATISTICS_BYTES: This attribute specifies the Ingress
Statistics Bytes of specific user.

EGRESS_STATISTICS_PACKETS: This attribute specifies the Egress
Statistics Packets of specific user.

EGRESS_STATISTICS_BYTES: This attribute specifies the Egress
Statistics Bytes of specific user.

5. Security Considerations

   None.

6. IANA Considerations

   None.

7. Normative References

   [I-D.gu-nfvrg-cloud-bng-architecture]
            Gu, R. and S. Hu, "Control and User Plane Separation
            Architecture of BNG", draft-gu-nfvrg-cloud-bng-
            architecture-01 (work in progress), July 2017.

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

   [RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group
            MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000,
            <https://www.rfc-editor.org/info/rfc2863>.

   [RFC5837] Atlas, A., Ed., Bonica, R., Ed., Pignataro, C., Ed., Shen,
            N., and JR. Rivers, "Extending ICMP for Interface and
            Next-Hop Identification", RFC 5837, DOI 10.17487/RFC5837,
            April 2010, <https://www.rfc-editor.org/info/rfc5837>.

Authors' Addresses

Michael Wang (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: wangzitao@huawei.com

Rong Gu
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing, Beijing 100053
China

Email: gurong_cmcc@outlook.com

Victor Lopez
Telefonica
Sur 3 building, 3rd floor, Ronda de la Comunicacion s/n
Madrid 28050
Spain

Email: victor.lopezalvarez@telefonica.com

Sujun Hu
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing, Beijing 100053
China

Email: shujun_hu@outlook.com