

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

S. Hares
Huawei
A. Dass
Ericsson
March 21, 2016

I2RS Data Flow Requirements
draft-hares-i2rs-dataflow-req-03.txt

Abstract

This document covers requests to the netmod and netconf Working Groups for functionality to support the data flows described in the I2RS architecture and the I2RS use cases requirements summary.

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 September 22, 2016.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Summary of I2RS Data Flow Requirements	3
3. Generic Interfaces to Routing Functions	5
3.1. I2RS-Generic Interface to Local-RIB	5
3.2. I2RS-Generic interfaces to Policies	5
3.3. I2RS Data Flow Requirements	5
4. Large Data Flow Requirements	5
4.1. Large Data Flow Use Case Requirements	6
4.1.1. Data Requirements Supported in pub-sub Requirements	7
4.1.2. Data Flow Requirements Outside of Pub/Sub Requirements	7
4.1.3. I2RS Data Flow Requirements	7
4.2. Traffic Flow Measurements	8
4.2.1. Protocol Requirements based on Traffic Flows	9
4.2.2. I2RS Data Flow Requirements	9
4.3. Action sequences in Data Models	10
4.3.1. Action sequences	10
4.3.2. TCAM use case	11
4.3.3. I2RS Data Flow Requirement	11
4.4. Operation during network outages or attacks	11
4.4.1. Periods of Network Outage	12
4.4.2. I2RS Data Flow Requirements	12
5. Changes to YANG	13
6. IANA Considerations	13
7. Security Considerations	13
8. Acknowledgements	13
9. References	13
9.1. Normative References	13
9.2. Informative References	14
Authors' Addresses	16

1. Introduction

The Interface to the Routing System (I2RS) Working Group is chartered with providing architecture and mechanisms to inject into and retrieve information from the routing system. The I2RS Architecture document [I-D.ietf-i2rs-architecture] abstractly documents a number of requirements for implementing the I2RS requirements.

The I2RS Working Group has chosen to use the YANG data modeling language [RFC6020] as the basis to implement its mechanisms.

Additionally, the I2RS Working group has chosen to use the NETCONF [RFC6241] and its similar but lighter-weight relative RESTCONF [I-D.ietf-netconf-restconf] as the protocols for carrying I2RS. NETCONF and RESTCONF are suitable for handling the configuration

portion of the I2RS protocol, but need extensions to handle the I2RS use cases described in [I-D.ietf-i2rs-usecase-reqs-summary]. The requirements for these functionalities include:

- o ephemeral state - as defined in [I-D.ietf-i2rs-ephemeral-state]
- o notifications and events - as defined in [I-D.ietf-i2rs-pub-sub-requirements]
- o traceability - as defined in [I-D.ietf-i2rs-traceability]
- o protocol security - as defined in [I-D.ietf-i2rs-protocol-security-requirements]
- o Generic interfaces to Protocol Local-RIBs or Policy Data bases,
- o Large data flows,
- o Traffic monitoring data,
- o Data flows for Action sequences, and
- o data flows during network outages or attacks

This document describes the protocol requirements for these last five types of requirements. The first section summarizes the data flow requirements gathered from the Use Cases. These list of requirements are being presented to the I2RS Working group to determine if the requirement is need for the first revision of the I2RS protocol. Future revisions may add additional data flow requirement. The authors have indicated their suggestion with the following abbreviation:

v1 - version 1, or

fv - future version

Section 2 details how the I2RS use case requirements for Generic interfaces to protocol RIBs or policy data base do not add any requirements to the I2RS protocol. Section 3 describes how the describes

2. Summary of I2RS Data Flow Requirements

Additional requirements from Generic Interface:None

The additional Data flow requirements are:

DF-REQ-01: Support writing to the ephemeral copy of the Local RIB with three different types of checks: minimal data reception checks (TLVs of data oacket valid), all non-referential checks (e.g. do not do leafref, MUST, instance identifiers), and do referential checks via three different rpcs. [v1]

DF-REQ-02: The support of large data transfers in a data format agnostic format. The NETCONF protocol now supports XML and JSON. I2RS protocol should also support other data formats (MTL [fv], raw ascii stream [fv])

DF-REQ-03: Support of I2RS Agent and I2RS Client negotiating specific transport and transport options out the options that are available (v1),

DF-REQ-04: Support for the ability to send traffic monitoring information using IPFIX protocol and IPFIX templates (fv),

(DF-REQ-05): Support of traffic statistics for filter-based policies (BGP-FS, I2RS FB-RIB, policy routing), IPPM, SFLOW, and others in yang data model format. (authors mixed - v1 or fv)

DF-REQ-06: I2RS should be able to support an action which allocates internal resources for the I2RS agent (memory, processing time, interrupts) and outbound data flow bandwidth. It is expected that an action would be included in a data model in an "rpc"-like format in yang. (v1)

DF-REQ-07: The I2RS should be able to support an action that interacts with routing OAM functions. [Editor: Operator-applied priorities and manual control must support limiting I2RS actions with OAM.] (v1 or v2)

DF-REQ-08: The I2RS Agent must be able signals that it will be using different protocol with different constraints (security, priority of data, or transport) or different constraints on the existing protocol (smaller message sizes, different priorities on data carried, or different security levels). (v2) [Editor's Note: Should this be for network outages or for just security attacks?]

DF-REQ-09: Yang MUST have a way to indicate in a data model has actions which allow: different transports, different resource constraints, or different security. (v1)

DF-REQ-10: Yang MUST have a way to indicate a data model has different levels of checking where: lowest level is message form only, medium level checks message format plus data syntax, and highest level uses the message format, data syntax and referential

check netconf configuration does. The default level for I2RS is message format plus data syntax. (v1)

3. Generic Interfaces to Routing Functions

The I2RS use case requirement suggests that a generic interface be created to protocol local RIBs and a generic interface be available to configure policies.

3.1. I2RS-Generic Interface to Local-RIB

The I2RS requirements ([I-D.ietf-i2rs-usecase-reqs-summary]) require that a generic interface be defined to the local-RIB in protocols. This type of data flow does not require a new type of data flow, but the definition of a new data model that creates a generic local RIB and has operations to funnel this generic Local-RIB to a specific protocol.

The Protocol Independent Use case (PI-REQ-11) Local RIB use case suggest the I2RS protocol has three levels of checks: minimal data reception checks (TLVs of data align), all non-referential checks (e.g. do not do leafref, MUST, instance identifiers), and do referential checks. This feature could be supported through different rpc calls to the LOCAL RIB.

3.2. I2RS-Generic interfaces to Policies

The I2RS requirements suggest that I2RS have a generic interface to routing policies for protocols, routing distribution, or routing protocols. This generic interface is currently being implemented as common definitions for data models. At this time, This generic interface does not need additional protocol requirements.

3.3. I2RS Data Flow Requirements

[DF-REQ-01] Support writing to the ephemeral copy of the Local RIB with three different types of checks: minimal data reception checks (TLVs of data oacket valid), all non-referential checks (e.g. do not do leafref, MUST, instance identifiers), and do referential checks via three different rpcs.

4. Large Data Flow Requirements

This section describes the data flow requirements for large data flows, traffic flows measurements, CDNI traffic flows, OAM and Action rgeuests, data flows during outages or network attacks (DDoS (Distributed Denial of Service) or other network attacks), and non-

secure data flows. These data flows are data flows which are not configuration based data flows.

4.1. Large Data Flow Use Case Requirements

The I2RS use case for Large Data Collection systems [I-D.ietf-i2rs-usecase-reqs-summary] requires the I2RS protocol and data models:

- o be able to be done at a high frequency and resolution with minimal impact to devices memory or CPU (L-Data-REQ-01) ,
- o use a data model which allows definition of the form as part of the data model (L-Data-REQ-02) ,
- o support a publication/subscription mechanism with push/pull mechanism (L-Data-REQ-03),
- o (supports capability negotiation for level of transport, security, and error handling as a general configurations, per I2RS client-agent protocol for all interfaces and all time instance, or per I2RS interface client-agent protocol per specific interface or per time instances. (L-Data-REQ-04,L-Data-REQ-06, L-Data-REQ-07, L-Data-REQ-08, and L-Data-REQ-09),
- o dynamic subscription model set-up via IPFIX (L-REQ-12c),
- o support of subscriber and consumer I2RS-Agent pairs (L-REQ-12d),
- o remapping of Node's databases,
- o data format agnostic (L-Data-REQ-05),
- o data models and I2RS protocol additions that support of query, introspection using data-base model that support a set of capabilities, data filters, and error handling (stale data, repeated transport failures, and other errors.) Introspection supports data verification, inclusion of legacy data, and merging of data flows based on meta-data. (L-Data-REQ-11, L-Data-REQ-13),
- o Support of push of data synchronously or asynchronously via registered subscriptions (L-Data-REQ-12a).
- o Pull of data in one-shot or multiple sequences (L-Data-REQ-12b), and
- o dynamic subscription model set-up via IPFIX Feed (?) (L-REQ-12c)

4.1.1. Data Requirements Supported in pub-sub Requirements

All use case requirements for the publication/subscription service for the push service from large data requirements 01-04 and 6-12 is found in [I-D.ietf-i2rs-pub-sub-requirements], and an example protocol addition to netconf is include in [I-D.ietf-netconf-yang-push].

The requirements for the publication/subscription service for the pull model are not specified in the [I-D.ietf-i2rs-pub-sub-requirements], but a majority of the pub-sub requirements and mechanisms can be reused. In a pull, the publisher prepares the data that is pulled by a few receivers who then distribute it to the receivers. The pull mechanism would have a different "pull latency" versus the push latency, and a set of parameters which indicate the amount of data stored if receivers did not pull the data within a certain time.

At this time, the pull-model of the publication/subscription model is not being requested by vendors or operators.

4.1.2. Data Flow Requirements Outside of Pub/Sub Requirements

The data flow requirements for large data flows also include support for data flows outside of publication/subscription via any transport (L-Dat-REQ-04) and any data format (L-Data-REQ-05). It is unclear whether the L-DATA-REQ-12 really wants to utilize IPFIX protocol or just IPFIX templates to handle the monitoring data.

Editor note: It becomes a question for the WG as to whether these are necessary for version 1 of the I2RS protocol, version 2 or never.

4.1.3. I2RS Data Flow Requirements

The following requirements are additional data flow requirements for large data flows.

(DF-REQ-02): Support of any data format including: XML, JSON, (MTL (Alias/WaveFormat,.mtl), protobufs, and ascii. NETCONF already supports the pub/sub push model with XML and JSON. It is important to determine what is needed.

DF-REQ-03: Support of I2RS Agent and I2RS Client negotiating specific transport and transport options out the options that are available,

(DF-REQ-04): support of the ability send information using IPFIX templates over the IPFIX protocol. [Note: This requirement is

unclear in the use case so it is included here to determine the working groups input. This would include I2RS protocol to have NETCONF/RESTCONF + IPFIX.]

[I-D.ietf-netconf-yang-push] supports XML and JSON in its first release, and provides an ability to register extra formats, but these requirements should also support large data flows sent outside of the publication-subscription service.

4.2. Traffic Flow Measurements

The I2RS requirements for the Protocol independent use cases requires the support off interactions with traffic flow and other network management Protocols (requirements PI-REQ-05, PI-REQ06) in [I-D.ietf-i2rs-usecase-reqs-summary]).

The following IETF protocol pass traffic related information:

- o BGP Flow Specification (BGP-FS) ([RFC5575]
- o IPFIX - IP Flow Information ([RFC7011]) that reports on a wide variety of routing system statistics, and
- o IPPM - IP Performance mangement ([RFC2330], [RFC7312]) that reports on one-way or two-way end-to-end network performance statistics,

In addition the SFLOW([RFC3176]) of layer 2 devices is supported by many routers. Other traffic flows may be measured in support of IDS/IPS, but these will be covered in the section on security flows.

Additional traffic flow models are being defined to configure traffic flow policy and to monitor the statistics on the use of the traffic flow statistics:

- o BGP Flow Specification (BGP-FS) yang model
[I-D.wu-idr-flowspec-yang-cfg] contains flow filter match statistics.
- o I2RS Filter-Based RIB yang model
([I-D.kini-i2rs-fb-rib-info-model],
[I-D.hares-i2rs-fb-rib-data-model])- yang model contains ephemeral flow statistics,
- o Filter-Based RIB (draft-hares-rtgwg-fb-rib-data-model) contains both flow filter match statistics,

4.2.1. Protocol Requirements based on Traffic Flows

Due to the potentially large data flow these statistics should be handle by push pub-sub model or a pull pub-sub model. Thresholds for data models may be passed by the event portion of the push/pull pub-sub model. The pub-sub model will allow the I2RS client-I2RS Agent to meter the amount of data flow these statistics carry. The push portion of the pub-sub model is supported by [I-D.ietf-netconf-yang-push], but the pull portion of the pub-sub model is not defined.

Alternatively I2RS can use the the IPFIX protocol ([RFC7011]) as a component protocol. I2RS processes can support an IPFIX exporting process sending data to a node to a node on a collector process. The IPFIX templates can be configured as ephemeral state or configuration state. The IPFIX data flows may run over SCTP, UDP, or TCP utilizing the congestion services at each time. The IPFIX connections assumes that: a) congestion is an temporary anomaly, b) dropping data during a congestion is reported, and c) for some exporting process it is acceptable to have drop data in a reliable protocol. The I2RS protocol must support the establishment of an IPFIX connection.

Traffic monitoring can occur in a network under DDoS with high levels of congestion and loss the use of these protocols which rely on transport-level retransmission may not be as resilient as needed for network security functions (NSF). These are considered in section 5 on operations during network outages or congestoin.

The Flow Filtering data models with policy rules (BGP Flow Specification, I2RS Filter-Based RIB, and n-tuple policy routing RIB) often track how often these policies are match. These statistics can also be pushed/pulled in a publication/subscription with yang data-model defined format or an IPFIX exporting process format. Similarly IPPM statistics or SFLOW data, be sent via publication/subscription service in yang data model format or in a IPFIX Template or as XML or JSON representation of a yang data model. These additional sources do not change the requirements for the push publication/subscription or expand the

Summary: The pub-sub model push or pull may have to support additional formats (E.g. SFLOW, IPFIX) as well as yang data models.

4.2.2. I2RS Data Flow Requirements

DF-REQ-04: Support for the ability to send traffic monitoring information using IPFIX protocol and IPFIX templates, [Editor: This requirement is unclear in the use case so this requirement is to

confirm the I2RS WG desire for NETCONF/RESTCONF + IPFIX == I2RS protocol]

(DF-REQ-05): Support of traffic statistics for filter-based policies (BGP-FS, I2RS FB-RIB, policy routing), IPPM, SFLOW, and others in yang data model format.

4.3. Action sequences in Data Models

This section considers the data flow requirements in sequences of actions (e.g. calculate topology and install), and actions that interact with TCAMs (e.g. putting filters in TCAMs).

4.3.1. Action sequences

Several of the I2RS requirements from the use cases require a sequence of events with the following actions:

1. query data in protocol independent model (topology, RIB, Filter-RIB), or protocol),
2. start calculation (or re-calculation) in protocol function,
3. Report results,
4. install topology or RIB calculated,
5. check results,
6. recycle.

The actions included looking for overlapping BGP routes, IGP LFA calculation, ECMP load balancing traffic, optimizing paths via MPLS-TE, CCNE re-optimization, and virtual topology creation.

An alternate pattern within the requirements is if the topology is calculated off-line, and uploaded.

These action patterns may involve an interaction of the I2RS action sequences with existing OAM functions in the routing system.

NETCONF/RESTCONF have the concepts of an "rpc" for a configuration enabled action, but these action sequences should have the ability to have the following characteristics:

- o the ability to request a reservation of resources for this effort so the action sequence does not start unless there is enough calculation or response bandwidth in a node,

- o the ability to have validation on off-line calculated data so this critical data does not have errors
- o the ability to "prioritize" notification or reports ahead of other I2RS data streams to allow process to work.

4.3.2. TCAM use case

Note: TCAM (hardware memory) in general is used in most of the routing devices for faster address lookup that enables fast routing. The TCAM also provides the flexibility to manually specify how much TCAM space you want to allocate to a specific feature or action (like routing, switching, security, ACL etc.). There may be cases when a manual allocation of memory (H/W or S/W) could restrict the I2RS functions. To allow operators the control they need, the manual allocation must be considered.

A few questions need to be considered:

- o how do we handle Action sequences or TCAM?
- o After the allocation internal resources, what shall be the timing or process to release those resources?
- o What should happen in case there are issues getting the internal resource allocation done? - Should we just send a event/error to the client or should something else happen?

4.3.3. I2RS Data Flow Requirement

I2RS-DF-REQ-06: I2RS should be able to support an action which allocates internal resources for the I2RS agent (memory, processing time, interrupts) and outbound data flow bandwidth. It is expected that an action would be included in a data model in an "rpc"-like format in yang.

DF-REQ-07: The I2RS should be able to support an action that interacts with routing OAM functions. [Editor: Operator-applied priorities and manual control must support limiting I2RS actions with OAM.]

4.4. Operation during network outages or attacks

The router needs dynamic management during periods of outage or periods of security attack.

4.4.1. Periods of Network Outage

During periods of outage, the I2RS protocol must operate when data bandwidth is reduced and network connectivity fluctuates. I2RS agents must be able to adjust operation of event notifications, logging, or data traffic during this period. Data Models and I2RS agent configuration must allow operator-applied policy to prioritize data during this period. The I2RS Agent should be able to signal the I2RS Client that such a time period is occurring.

Some periods of outage are caused by security attacks (DDoS or target incident that exploit vulnerabilities in software, network devices, protocols.) [I-D.hares-i2nsf-mgtflow-reqs] provides a description of the data flow needed from network security controllers to the network security devices or firewalls in routers. Editor's Note: I2NSF is reviewing this draft, and will give feedback on this requirement.

Network Outages may occur due to several issues including the security reasons but network downtimes caused by security reasons may also be quite diverse, for example a network outage due to DDoS attack, botnets, malware attack, identity theft, or incidents that incident that exploit vulnerabilities in software, network devices, protocols. And if an outage has occurred due to security reasons then other safeguard measures could overlap with the I2RS based prioritization.

Since these outages can overlap with the network security controllers using I2NSF protocol to contact the network service functions (NSF) or virtual service functions (vNSF), the I2NSF working group should be consulted to determine what I2RS versus I2NSF needs are, and what conflicts exist. [I-D.hares-i2nsf-mgtflow-reqs] provides a description of the data flow needed from network security controllers to the network security devices or firewalls in routers. The I2NSF WG is reviewing this draft.

Editor's note: Do we want a general OAM feature or a feature specific to DDoS and security attack case?

4.4.2. I2RS Data Flow Requirements

DF-REQ-08: The I2RS Agent must be able signals that it will be using different protocol with different constraints (security, priority of data, or transport) or different constraints on the existing protocol (smaller message sizes, different priorities on data carried, or different security levels). [Editor's Note: Should this be for network outages or for just security attacks?]

5. Changes to YANG

To support the above requirements, the yang modules will need to support the following features:

- o DF-REQ-09: Yang MUST have a way to indicate in a data model has actions which allow: different transports, different resource constraints, or different security.
- o DF-REQ-10: Yang MUST have a way to indicate a data model has different levels of checking where: lowest level is message form only, medium level checks message format plus data syntax, and highest level uses the message format, data syntax and referential check netconf configuration does. The default level for I2RS is message format plus data syntax.

6. IANA Considerations

There are no IANA requirements for this document.

7. Security Considerations

The security requirements for the I2RS protocol are covered in [I-D.ietf-i2rs-protocol-security-requirements] document.

8. Acknowledgements

The following people have aided in the discuss

- o Russ White,
- o Joel Halpern,
- o Linda Dunbar,
- o Frank Xia, and
- o Robert Moskowitz

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

9.2. Informative References

- [I-D.hares-i2nsf-mgtflow-reqs]
Hares, S., "I2NSF Data Flow Requirements", draft-hares-i2nsf-mgtflow-reqs-00 (work in progress), March 2016.
- [I-D.hares-i2rs-fb-rib-data-model]
Hares, S., Kini, S., Dunbar, L., Krishnan, R., Bogdanovic, D., and R. White, "Filter-Based RIB Data Model", draft-hares-i2rs-fb-rib-data-model-02 (work in progress), February 2016.
- [I-D.ietf-dots-requirements]
Mortensen, A., Moskowitz, R., and T. Reddy, "DDoS Open Threat Signaling Requirements", draft-ietf-dots-requirements-00 (work in progress), October 2015.
- [I-D.ietf-i2nsf-problem-and-use-cases]
Hares, S., Dunbar, L., Lopez, D., Zarny, M., and C. Jacquenet, "I2NSF Problem Statement and Use cases", draft-ietf-i2nsf-problem-and-use-cases-00 (work in progress), February 2016.
- [I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-13 (work in progress), February 2016.
- [I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-04 (work in progress), March 2016.
- [I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-03 (work in progress), March 2016.
- [I-D.ietf-i2rs-pub-sub-requirements]
Voit, E., Clemm, A., and A. Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-05 (work in progress), February 2016.
- [I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-08 (work in progress), October 2015.

- [I-D.ietf-i2rs-traceability]
Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", draft-ietf-i2rs-traceability-07 (work in progress), February 2016.
- [I-D.ietf-i2rs-usecase-reqs-summary]
Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", draft-ietf-i2rs-usecase-reqs-summary-02 (work in progress), March 2016.
- [I-D.ietf-mile-rfc5070-bis]
Danyliw, R., "The Incident Object Description Exchange Format v2", draft-ietf-mile-rfc5070-bis-16 (work in progress), February 2016.
- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-10 (work in progress), March 2016.
- [I-D.ietf-netconf-yang-push]
Clemm, A., Prieto, A., Voit, E., Tripathy, A., and E. Einar, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-01 (work in progress), February 2016.
- [I-D.kini-i2rs-fb-rib-info-model]
Kini, S., Hares, S., Dunbar, L., Ghanwani, A., Krishnan, R., Bogdanovic, D., and R. White, "Filter-Based RIB Information Model", draft-kini-i2rs-fb-rib-info-model-03 (work in progress), February 2016.
- [I-D.wu-idr-flowspec-yang-cfg]
Wu, N., Zhuang, S., and A. Choudhary, "A YANG Data Model for Flow Specification", draft-wu-idr-flowspec-yang-cfg-02 (work in progress), October 2015.
- [RFC2330] Paxson, V., Almes, G., Mahdavi, J., and M. Mathis, "Framework for IP Performance Metrics", RFC 2330, DOI 10.17487/RFC2330, May 1998, <<http://www.rfc-editor.org/info/rfc2330>>.
- [RFC3176] Phaal, P., Panchen, S., and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", RFC 3176, DOI 10.17487/RFC3176, September 2001, <<http://www.rfc-editor.org/info/rfc3176>>.

- [RFC5070] Danyliw, R., Meijer, J., and Y. Demchenko, "The Incident Object Description Exchange Format", RFC 5070, DOI 10.17487/RFC5070, December 2007, <<http://www.rfc-editor.org/info/rfc5070>>.
- [RFC5575] Marques, P., Sheth, N., Raszuk, R., Greene, B., Mauch, J., and D. McPherson, "Dissemination of Flow Specification Rules", RFC 5575, DOI 10.17487/RFC5575, August 2009, <<http://www.rfc-editor.org/info/rfc5575>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <<http://www.rfc-editor.org/info/rfc7011>>.
- [RFC7312] Fabini, J. and A. Morton, "Advanced Stream and Sampling Framework for IP Performance Metrics (IPPM)", RFC 7312, DOI 10.17487/RFC7312, August 2014, <<http://www.rfc-editor.org/info/rfc7312>>.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: August 12, 2016

S. Hares
Q. Wu
Huawei
R. White
Ericsson
February 9, 2016

Filter-Based Packet Forwarding ECA Policy
draft-hares-i2rs-pkt-eca-data-model-02.txt

Abstract

This document describes the yang data model for packet forwarding policy that filters received packets and forwards (or drops) the packets. Prior to forwarding the packets out other interfaces, some of the fields in the packets may be modified. If one considers the packet reception an event, this packet policy is a minimalistic Event-Match Condition-Action policy. This policy controls forwarding of packets received by a routing device on one or more interfaces on which this policy is enabled. The policy is composed of an ordered list of policy rules. Each policy rule contains a set of match conditions that filters for packets plus a set of actions to modify the packet and forward packets. The match conditions can match tuples in multiple layers (L1-L4, application), interface received on, and other conditions regarding the packet (size of packet, time of day). The modify packet actions allow for setting things within the packet plus decapsulation and encapsulation packet. The forwarding actions include forwarding via interfaces, tunnels, or nexthops and dropping the packet. The policy model can be used with the session ephemeral (BGP Flow Specifications), reboot ephemeral state (I2RS ephemeral), and non-ephemeral routing/forwarding state (e.g. configuration state).

Status of This Memo

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

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

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

This Internet-Draft will expire on August 12, 2016.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Definitions and Acronyms	3
1.2. Antecedents this Policy in IETF	3
2. Generic Route Filters/Policy Overview	4
3. BNP Rule Groups	5
4. BNP Generic Info Model in High Level Yang	7
5. i2rs-eca-policy Yang module	10
6. IANA Considerations	35
7. Security Considerations	35
8. Informative References	36
Authors' Addresses	37

1. Introduction

This document describes the yang data model for packet forwarding policy that filters received packets and forwards (or drops) the packets. Prior to forwarding the packets out other interfaces, some of the fields in the packets may be modified. If one considers the reception of a packet as an event, this minimalistic Event-Match Condition-Action policy. If one considers the reception of packets containing Layer 1 to Layer 4 + application data a single packet, then this minimalistic policy can be called a packet-only ECA policy. This document will use the term packet-only ECA policy for this model utilizing the term "packet" in this fashion.

This packet-only ECA policy data model supports an ordered list of ECA policy rules where each policy rule has a name. The match condition filters include matches on

- o packet headers for layer 1 to layer 4,
- o application protocol data and headers,
- o interfaces the packet was received on,
- o time packet was received, and
- o size of packet.

The actions include packet modify actions and forwarding options. The modify options allow for the following:

- o setting fields in the packet header at Layer 2 (L2) to Layer 4 (L4), and
- o encapsulation and decapsulation the packet.

The forwardingng actions allow forwardsing the packet via interfaces, tunnels, next-hops, or dropping the packet. setting things within the packet at Layer 2 (L2) to layer 4 (L4) plus overlay or application data.

The first section of this draft contains an overview of the policy structure. The second provides a high-level yang module. The third contains the yang module.

1.1. Definitions and Acronyms

INSTANCE: Routing Code often has the ability to spin up multiple copies of itself into virtual machines. Each Routing code instance or each protocol instance is denoted as Foo_INSTANCE in the text below.

NETCONF: The Network Configuration Protocol

PCIM - Policy Core Information Model

RESTconf - http programmatic protocol to access yang modules

1.2. Antecedents this Policy in IETF

Antecedents to this generic policy are the generic policy work done in PCIM WG. The PCIM work contains a Policy Core Information Model (PCIM) [RFC3060], Policy Core Informational Model Extensions [RFC3460] and the Quality of Service (QoS) Policy Information Model (QPIM) ([RFC3644]) From PCIM comes the concept that policy rules which are combined into policy groups. PCIM also refined a concept

of policy sets that allowed the nesting and aggregation of policy groups. This generic model did not utilize the concept of sets of groups, but could be expanded to include sets of groups in the future.

2. Generic Route Filters/Policy Overview

This generic policy model represents filter or routing policies as rules and groups of rules.

The basic concept are:

Rule Group

A rule group is is an ordered set of rules .

Rule

A Rule is represented by the semantics "If Condition then Action".
A Rule may have a priority assigned to it.

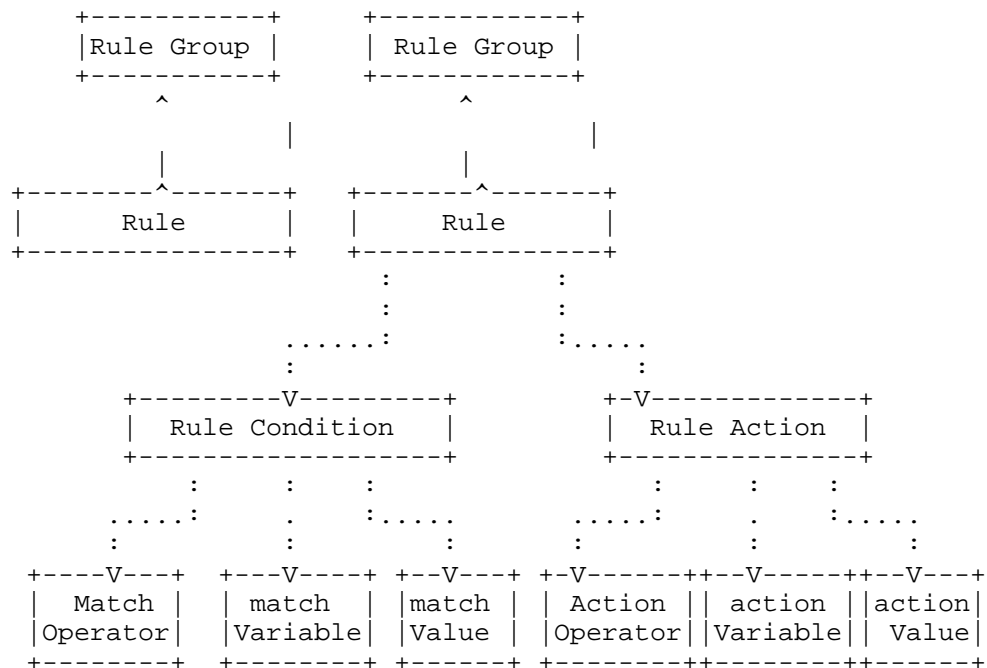


Figure 1: ECA rule structure

3. BNP Rule Groups

The pkt ECA policy is an order set of pkt-ECA policy rules. The rules assume the event is the reception of a packet on the machine on a set of interfaces. This policy is associated with a set of interfaces on a routing device (physical or virtual).

A Rule group allows for the easy combination of rules for management stations or users. A Rule group has the following elements:

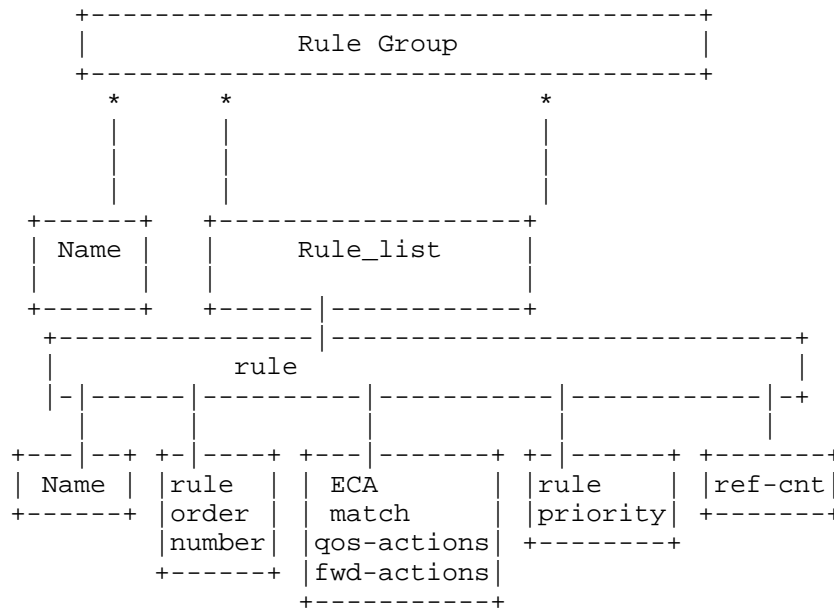
- o name that identifies the grouping of policy rules
- o module reference - reference to a yang module(s) in the yang module library that this group of policy writes policy to
- o list of rules

Rule groups may have multiple policy groups at specific orders. For example, policy group 1 could have three policy rules at rule order 1 and four policy rules at rule order 5.

The rule has the following elements: name, order, status, priority, reference cnt, and match condition, and action as shown in figure 2. The order indicates the order of the rule within the complete list. The status of the rule is (active, inactive). The priority is the priority within a specific order of policy/filter rules. A reference count (refcnt) indicates the number of entities (E.g. network modules) using this policy. The generic rule match-action conditions have match operator, a match variable and a match value. The rule actions have an action operator, action variable, and an action value.

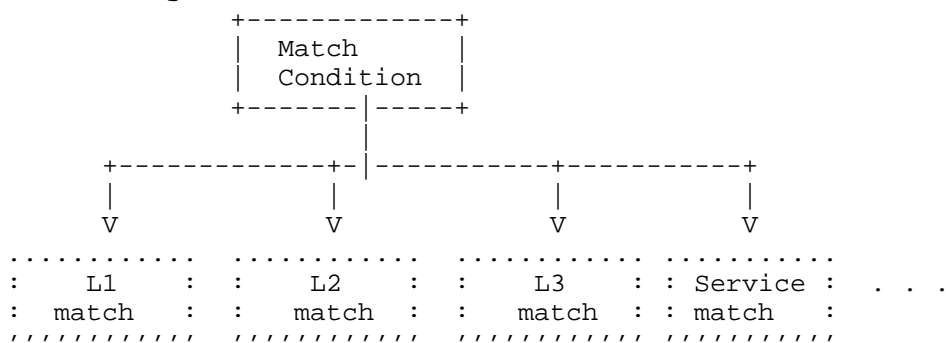
Rules can exist with the same rule order and same priority. Rules with the same rule order and same priority are not guaranteed to be at any specific ordering. The order number and priority have sufficient depth that administrators who wish order can specify it.

Figure 2 - Rule Group



The generic match conditions are specific to a particular layer are refined by matches to a specific layer (as figure 3 shows), and figure 5's high-level yang defines. The general actions may be generic actions that are specific to a particular layer (L1, L2, L3, service layer) or time of day or packet size. The qos actions can be setting fields in the packet at any layer (L1-L4, service) or encapsulating or decapsulating the packet at a layer. The fwd-actions are forwarding functions that forward on an interface or to a next-hop. The rule status is the operational status per rule.

Figure 3



4. BNP Generic Info Model in High Level Yang

Below is the high level inclusion

Figure 5

```
module:bnp-eca-policy
import ietf-inet-types {prefix "inet"}
import ietf-interface {prefix "if"}
import ietf-i2rs-rib {prefix "i2rs-rib"}

    import ietf-interfaces {
    prefix "if";
    }
    import ietf-inet-types {
    prefix inet;
    //rfc6991
    }

    import ietf-i2rs-rib {
    prefix "i2rs-rib";
```

Below is the high level yang diagram


```

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

```

The three levels of policy are expressed as:

Config Policy definitions

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

Operational State for Policy

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

figure

The filter matches struture is shown below

```
module:i2rs-pkt-eca-policy
  +--rw pkt-eca-policy-cfg
  |   +--rw pkt-eca-policy-set
  |   |   +--rw groups* [group-name]
  |   |   |   ...
  |   +--rw rules [order-id rule-name]
  |       +--rw eca-matches
  |       |   +--case: interface-match
  |       |   +--case: L1-header-match
  |       |   +--case: L2-header-match
  |       |   +--case: L3-header-match
  |       |   +--case: L4-header-match
  |       |   +--case: Service-header-match
  |       |   +--case: packet-size
  |       |   +--case: time-of-day
```

```

module:i2rs-pkt-eca-policy
  +--rw pkt-eca-policy-cfg
  |   +--rw pkt-eca-policy-set
  |   |   +--rw groups* [group-name]
  |   |   |   ...
  |   |   +--rw rules* [order-id rule-name]
  |   |   |   +--rw eca-matches
  |   |   |   |   . . .
  |   |   |   +--rw ecq-qos-actions
  |   |   |   |   +--rw cnt-actions
  |   |   |   |   +--rw mod-actions
  |   |   |   |   |   +--case interface-actions
  |   |   |   |   |   +--case L1-action
  |   |   |   |   |   +--case L2-action
  |   |   |   |   |   +--case L3-action
  |   |   |   |   |   +--case L4-action
  |   |   |   |   |   +--case service-action
  |   |   |   +--rw eca-fwd-actions
  |   |   |   |   +--rw num-fwd-actions
  |   |   |   |   +--rw fwd-actions
  |   |   |   |   |   +--rw interface interface-ref
  |   |   |   |   |   +--rw next-hop rib-nexthop-ref
  |   |   |   |   |   +--rw route-attributes
  |   |   |   |   |   +--rw rib-route-attributes-ref
  |   |   |   |   |   +--rw fb-std-drop

```

5. i2rs-eca-policy Yang module

<CODE BEGINS> file "ietf-pkt-eca-policy@2016-02-09.yang"

```

module ietf-pkt-eca-policy {
  namespace "urn:ietf:params:xml:ns:yang:ietf-pkt-eca-policy";
  // replace with iana namespace when assigned
  prefix "pkt-eca-policy";

  import ietf-routing {
    prefix "rt";
  }
  import ietf-interfaces {
    prefix "if";
  }
  import ietf-inet-types {
    prefix inet;
    //rfc6991
  }

  import ietf-i2rs-rib {

```

```
    prefix "i2rs-rib";
  }

// meta
  organization "IETF I2RS WG";

contact
  "email: shares@ndzh.com
    email: russ.white@riw.com
    email: linda.dunbar@huawei.com
    email: bill.wu@huawei.com";

description
  "This module describes a basic network policy
    model with filter per layer.";

  revision "2016-02-09" {
    description "initial revision";
    reference "draft-hares-i2rs-pkt-eca-policy-dm-00";
  }

// interfaces - no identity matches

// L1 header match identities
  identity l1-header-match-type {
    description
      " L1 header type for match ";
  }

identity l1-hdr-sonet-type {
  base l1-header-match-type;
  description
    " L1 header sonet match ";
}

identity l1-hdr-OTN-type {
  base l1-header-match-type;
  description
    " L1 header OTN match ";
}

  identity l1-hdr-dwdm-type {
    base l1-header-match-type;
    description
      " L1 header DWDM match ";
  }
```

```
    // L2 header match identities
    identity l2-header-match-type {
    description
    " l2 header type for match ";
    }

    identity l2-802-1Q {
    base l2-header-match-type;
    description
    " l2 header type for 802.1Q match ";
    }

    identity l2-802-11 {
    base l2-header-match-type;
    description
    " l2 header type for 802.11 match ";
    }

    identity l2-802-15 {
    base l2-header-match-type;
    description
    " l2 header type for 802.15 match ";
    }

    identity l2-NVGRE {
    base l2-header-match-type;
    description
    " l2 header type for NVGRE match ";
    }

    identity l2-mpls {
    base l2-header-match-type;
    description
    " l2 header type for MPLS match ";
    }

    identity l2-VXLAN {
    base l2-header-match-type;
    description
    " l2 header type for VXLAN match ";
    }

    // L3 header match identities
    identity l3-header-match-type {
    description
    " l3 header type for match ";
    }
```

```
identity l3-ipv4-hdr {
  base l3-header-match-type;
  description
" l3 header type for IPv4 match ";
}

identity l3-ipv6-hdr {
  base l3-header-match-type;
  description
" l3 header type for IPv6 match ";
}

identity l3-gre-tunnel {
  base l3-header-match-type;
description
" l3 header type for GRE tunnel match ";
}

// L4 header match identities

identity l4-header-match-type {
  description "L4 header
match types. (TCP, UDP,
SCTP, etc. )";
}

identity l4-tcp-header {
  base l4-header-match-type;
description "L4 header for TCP";
}

identity l4-udp-header {
  base l4-header-match-type;
  description "L4 header match for UDP";
}

identity l4-sctp-header {
  base l4-header-match-type;
  description "L4 header match for SCTP";
}

// Service header identities

identity service-header-match-type {
  description "service header
match types: service function path
(sf-path), SF-chain, sf-discovery,
and others (added here)";
```

```
    }

    identity sf-chain-meta-match {
      base service-header-match-type;
      description "service header match for
        meta-match header";
    }

    identity sf-path-meta-match {
      base service-header-match-type;
      description "service header match for
        path-match header";
    }

    identity rule-status-type {
description "status
  values for rule: invalid (0),
  valid (1), valid and installed (2)";
    }

    identity rule-status-invalid {
base rule-status-type;
      description "invalid rule status.";
    }

    identity rule-status-valid {
      base rule-status-type;
      description "This status indicates
        a valid rule.";
    }

    identity rule-status-valid-installed {
      base rule-status-type;
      description "This status indicates
        an installed rule.";
    }

    identity rule-status-valid-inactive {
      base rule-status-type;
      description "This status indicates
        a valid ruled that is not installed.";
    }

    grouping interface-match {
      leaf match-if-name {
        type if:interface-ref;
        description "match on interface name";
      }
    }
```

```
    description "interface
    has name, description, type, enabled
    as potential matches";
}

grouping interface-actions {
  description
  "interface action up/down and
  enable/disable";
  leaf interface-up {
    type boolean;
    description
    "action to put interface up";
  }
  leaf interface-down {
    type boolean;
    description
    "action to put interface down";
  }
  leaf interface-enable {
    type boolean;
    description
    "action to enable interface";
  }
  leaf interface-disable {
    type boolean;
    description
    "action to disable interface";
  }
}

grouping L1-header-match {
  choice l1-header-match-type {
    case l1-hdr-sonet-type {
      // sonet matches
    }
    case L1-hdr-OTN-type {
      // OTN matches
    }
    case L1-hdr-dwdm-type {
      // DWDM matches
    }
  }
  description
  "The Layer 1 header match choices";
}
description
"The Layer 1 header match includes
```



```
        any reference to L1 technology";
    }

    grouping L1-header-actions {
        leaf l1-hdr-sonet-act {
            type uint8;
            description "sonet actions";
        }
        leaf l1-hdr-OTN-act {
            type uint8;
            description "OTN actions";
        }
        leaf l1-hdr-dwdm-act {
            type uint8;
            description "DWDM actions";
        }
        description "L1 header match
        types";
    }

    grouping L2-802-1Q-header {
        description
            "This is short-term 802.1 header
            match which will be replaced
            by reference to IEEE yang when
            it arrives. Qtag 1 is 802.1Q
            Qtag2 is 802.1AD";

        leaf vlan-present {
            type boolean;
            description " Include VLAN in header";
        }
        leaf qtag1-present {
            type boolean;
            description " This flag value indicates
            inclusion of one 802.1Q tag in header";
        }
        leaf qtag2-present{
            type boolean;
            description "This flag indicates the
            inclusion of second 802.1Q tag in header";
        }

        leaf dest-mac {
            type uint64; //change to uint48
            description "IEEE destination MAC value
            from the header";
        }
    }
}
```

```
    leaf src-mac {
        type uint64;                //change to uint48
        description "IEEE source MAC
            from the header";
    }
    leaf vlan-tag {
        type uint16;
        description "IEEE VLAN Tag
            from the header";
    }
    leaf qtag1 {
        type uint32;
        description "Qtag1 value
            from the header";
    }
    leaf qtag2 {
        type uint32;
        description "Qtag1 value
            from the header";
    }
    leaf L2-ethertype {
        type uint16;
        description "Ether type
            from the header";
    }
}

grouping L2-VXLAN-header {
    container vxlan-header {
        uses i2rs-rib:ipv4-header;
        leaf vxlan-network-id {
            type uint32;
            description "VLAN network id";
        }
        description " choices for
            L2-VLAN header matches.
            Outer-header only.
            Need to fix inner header. ";
    }
    description
        "This VXLAN header may
        be replaced by actual VXLAN yang
        module reference";
}

grouping L2-NVGRE-header {
```

```
    container nvgre-header {
        uses L2-802-1Q-header;
        uses i2rs-rib:ipv4-header;
        leaf gre-version {
            type uint8;
            description "L2-NVGRE GRE version";
        }
        leaf gre-proto {
            type uint16;
            description "L2-NVGRE protocol value";
        }
        leaf virtual-subnet-id {
            type uint32;
            description "L2-NVGRE subnet id value";
        }
        leaf flow-id {
            type uint16;
            description "L2-NVGRE Flow id value";
        }
        // uses L2-802-1Q-header;
        description
            "This NVGRE header may
            be replaced by actual NVGRE yang
            module reference";
    }
    description "Grouping for
        L2 NVGRE header."
}

grouping L2-header-match {
    choice l2-header-match-type {
        case l2-802-1Q {
            uses L2-802-1Q-header;
        }
        case l2-802-11 {
            // matches for 802.11 headers
        }
        case l2-802-15 {
            // matches for 802.1 Ethernet
        }
        case l2-NVGRE {
            // matches for NVGRE
            uses L2-NVGRE-header;
        }
        case l2-VXLAN-header {
            uses L2-VXLAN-header;
        }
    }
}
```

```
        }
        case l2-mppls-header {
            uses i2rs-rib:mppls-header;
        }
        description "Choice of L2
            headers for L2 match";
    }
description
    " The layer 2 header match includes
        any reference to L2 technology";
}

grouping L2-NVGRE-mod-acts {
    // actions for NVGRE
    leaf set-vsidi {
type boolean;
        description
            "Boolean flag to set VSID in packet";
    }
    leaf set-flowid {
        type boolean;
        description
            "Boolean flag to set VSID in packet";
    }
    leaf vsi {
        type uint32;
        description
            "VSID value to set in packet";
    }
    leaf flow-id {
        type uint16;
        description
            "flow-id value to set in packet";
    }
    description "L2-NVRE Actions";
}

grouping L2-VXLAN-mod-acts {
    leaf set-network-id {
        type boolean;
        description
            "flag to set network id in packet";
    }
    leaf network-id {
        type uint32;
        description
            "network id value to set in packet";
    }
}
```

```
        description "VXLAN header
        modification actions.";
    }

    grouping L2-mpls-mod-acts {
        leaf pop {
            type boolean;
            description
                "Boolean flag to pop mpls header";
        }
        leaf push {
            type boolean;
            description
                "Boolean flag to push value into
                mpls header";
        }
        leaf mpls-label {
            type uint32;
            description
                "mpls label to push in header";
        }
        description "MPLS modify
        header actions";
    }

    grouping l2-header-mod-actions {
        leaf l2-802-1Q {
            type uint8;
            description "actions for 802.1Q";
        }
        leaf l2-802-11 {
            type uint8;
            description "actions for 802.11";
        }
        leaf l2-802-15 {
            type uint8;
            description "actions for 802.15";
        }
    }

    uses L2-NVGRE-mod-acts;
uses L2-VXLAN-mod-acts;
    uses L2-mpls-mod-acts;

    description
        " The layer 2 header match includes
        any reference to L2 technology";
}
```

```
grouping L3-header-match {
    choice L3-header-match-type {
        case l3-ipv4-hdr {
            uses i2rs-rib:ipv4-header;
        }
        case l3-ipv6-hdr {
            uses i2rs-rib:ipv6-header;
        }
        case L3-gre-tunnel {
            uses i2rs-rib:gre-header;
        }
        description "match for L3
                    headers for IPv4, IPv6,
                    and GRE tunnels";
    }
    description "match for L3 headers";
}

grouping ipv4-encapsulate-gre {
    leaf encapsulate {
        type boolean;
        description "flag to encapsulate headers";
    }
    leaf ipv4-dest-address {
        type inet:ipv4-address;
        description "Destination Address for GRE header";
    }
    leaf ipv4-source-address {
        type inet:ipv4-address;
        description "Source Address for GRE header";
    }
    description "encapsulation actions for IPv4 headers";
}

grouping L3-header-actions {
    choice l3-header-act-type {
        case l3-ipv4-hdr {
            leaf set-ttl {
                type boolean;
                description "flag to set TTL";
            }
            leaf set-dscp {
                type boolean;
                description "flag to set DSCP";
            }
            leaf ttl-value {
                type uint8;
            }
        }
    }
}
```

```
        description "TTL value to set";
    }
    leaf dscp-val {
type uint8;
        description "dscp value to set";
    }
}
case l3-ipv6-hdr {
    leaf set-next-header {
        type boolean;
        description
            "flag to set next routing
            header in IPv6 header";
    }
    leaf set-traffic-class {
        type boolean;
        description
            "flag to set traffic class
            in IPv6 header";
    }
    leaf set-flow-label {
        type boolean;
        description
            "flag to set flow label
            in IPv6 header";
    }
    leaf set-hop-limit {
        type boolean;
        description "flag
            to set hop limit in
            L3 packet";
    }
    leaf ipv6-next-header {
        type uint8;
        description "value to
            set in next IPv6 header";
    }
    leaf ipv6-traffic-class {
        type uint8;
        description "value to set
            in traffic class";
    }
    leaf ipv6-flow-label {
        type uint16;
        description "value to set
            in IPv6 flow label";
    }
}
```

```
    }
    leaf ipv6-hop-limit {
        type uint8;
        description "value to set
            in hop count";
    }
}

case L3-gre-tunnel {
    leaf decapsulate {
        type boolean;
        description "flag to
            decapsulate GRE packet";
    }
    description "GRE tunnel
        actions" ;
}
description "actions that can
    be performed on L3 header";
}
description "actions to
    be performed on L3 header";
}

grouping tcp-header-match {
    leaf tcp-src-port {
        type uint16;
        description "source port match value";
    }
    leaf tcp-dst-port {
        type uint16;
        description "dest port value
            to match";
    }
    leaf sequence-number {
        type uint32;
        description "sequence number
            value to match";
    }
    leaf ack-number {
        type uint32;
        description "action value to
            match";
    }
    description "match for TCP
        header";
}
```



```
grouping tcp-header-action {
  leaf set-tcp-src-port {
    type boolean;
    description "flag to set
      source port value";
  }
  leaf set-tcp-dst-port {
    type boolean;
    description "flag to set source port value";
  }

  leaf tcp-s-port {
    type uint16;
    description "source port match value";
  }
  leaf tcp-d-port {
    type uint16;
    description "dest port value
      to match";
  }
  leaf seq-num {
    type uint32;
    description "sequence number
      value to match";
  }
  leaf ack-num {
    type uint32;
    description "action value to
      match";
  }
  description "Actions to
    modify TCP header";
}

grouping udp-header-match {
  leaf udp-src-port {
    type uint16;
    description "UDP source
      port match value";
  }
  leaf udp-dst-port {
    type uint16;
    description "UDP Destination
      port match value";
  }
  description "match values for
    UDP header";
}
```

```
}

grouping udp-header-action {
  leaf set-udp-src-port {
    type boolean;
    description "flag to set
      UDP source port match value";
  }
  leaf set-udp-dst-port {
    type boolean;
    description
      "flag to set UDP destination port match value";
  }
  leaf udp-s-port {
    type uint16;
    description "UDP source
      port match value";
  }
  leaf udp-d-port {
    type uint16;
    description "UDP Destination
      port match value";
  }

  description "actions to set
    values in UDP header";
}

grouping sctp-chunk {
  leaf chunk-type {
    type uint8;
    description "sctp chunk type value";
  }
  leaf chunk-flag {
    type uint8;
    description "sctp chunk type
      flag value";
  }
}

  leaf chunk-length {
    type uint16;
    description "sctp chunk length";
  }

  leaf chunk-data-byte-zero {
    type uint32;
    description "byte zero of
      stcp chunk data";
  }
}
```

```
    }
    description "sctp chunk
        header match fields";
}

grouping sctp-header-match {
    uses sctp-chunk;
    leaf stcp-src-port {
        type uint16;
        description "sctp header match
            source port value";
    }
    leaf sctp-dst-port {
        type uint16;
        description "sctp header match
            destination port value";
    }
    leaf sctp-verify-tag {
        type uint32;
        description "sctp header match
            verification tag value";
    }
    description "SCTP header
        match values";
}

grouping sctp-header-action {
    leaf set-stcp-src-port {
        type boolean;
        description "set source port in sctp header";
    }
    leaf set-stcp-dst-port {
        type boolean;
        description "set destination port in sctp header";
    }
    leaf set-stcp-chunk1 {
        type boolean;
        description "set chunk value in sctp header";
    }
    leaf chunk-type-value {
        type uint8;
        description "sctp chunk type value";
    }
    leaf chunk-flag-value {
        type uint8;
        description "sctp chunk type
            flag value";
    }
}
```

```
        leaf chunk-len {
            type uint16;
            description "sctp chunk length";
        }

        leaf chunk-data-bzero {
            type uint32;
            description "byte zero of
                sctp chunk data";
        }
        description "sctp qos actions";
    }

    grouping L4-header-match {
        choice l4-header-match-type {
            case l4-tcp-header {
                uses tcp-header-match;
            }
            case l4-udp-header {
                uses udp-header-match;
            }
            case l4-sctp {
                uses sctp-header-match;
            }
        }
        description "L4 match
            header choices";
    }
    description "L4 header
        match type";
}

grouping L4-header-actions {
    uses tcp-header-action;
    uses udp-header-action;
    uses sctp-header-action;
    description "L4 header matches";
}

grouping service-header-match {
    choice service-header-match-type {
        case sf-chain-meta-match {
            description "uses
                sfc-sfc:service-function-chain-grouping:
                + sfc-sfc:service-function-chain";
        }
        case sf-path-meta-match {
```

```
        description "uses
            sfc-spf:service-function-paths:
            + sfc-spf:service-function-path";
    }
    description "SFC header match
    choices";
}
description "SFC header and path
    matches";
}

grouping sfc-header-actions {
    choice service-header-match-type {
        case sf-chain-meta-match {
            leaf set-chain {
                type boolean;
                description "flag to set
                    chain in sfc. Should
                    be amended to use SFC service
                    chain matching.
                    uses sfc-sfc:service-function-chain-grouping:
                    + sfc-sfc:service-function-chain";
            }
        }
        case sf-path-meta-match {
            leaf set-path {
                type boolean;
                description "flag to set path in
                    sfc header. Amend to use sfc-spf
                    function headers. Uses
                    sfc-spf:service-function-paths:
                    + sfc-spf:service-function-path.";
            }
        }
    }
    description "choices in SFC for
        chain match and path match.";
}
description "modify action for
    SFC header.";
}

grouping rule_status {
    leaf rule-status {
        type string;
        description "status information
            free form string.";
    }
}
```

```
    leaf rule-inactive-reason {
        type string;
        description "description of
            why rule is inactive";
    }
    leaf rule-install-reason {
        type string;
        description "response on rule installed";
    }
    leaf rule-installer {
        type string;
        description "client id of installer";
    }
    leaf refcnt {
        type uint16;
        description "reference count on rule. ";
    }
    description
        "rule operational status";
}

// group status
grouping groups-status {
    list group_opstate {
        key "grp-name";
        leaf grp-name {
            type string;
            description "eca group name";
        }
    }
    leaf rules-installed {
        type uint32;
        description "rules in
            group installed";
    }
    list rules_status {
        key "rule-name";
        leaf rule-name {
            type string;
            description "name of rule ";
        }
        leaf rule-order {
            type uint32;
            description "rule-order";
        }
        description "rules per
            group";
    }
    description "group operational
```

```
        status";
    }
    description "group to rules
        list";
}

// links between rule to group

grouping rule-group-link {
    list rule-group {
        key rule-name;
        leaf rule-name {
            type string;
            description "rule name";
        }
        leaf group-name {
            type string;
            description "group name";
        }
        description "link between
            group and link";
    }
    description "rule-name to
        group link";
}

// rule status by name
grouping rules_opstate {
    list rules_status {
        key "rule-order rule-name";
        leaf rule-order {
            type uint32;
            description "order of rules";
        }
        leaf rule-name {
            type string;
            description "rule name";
        }
    }
    uses rule_status;
    description "eca rule list";
}
description "rules
    operational state";
}

// rule statistics by name and order
grouping rules_opstats {
    list rule-stat {
```

```
key "rule-order rule-name";
leaf rule-order {
  type uint32;
  description "order of rules";
}
leaf rule-name {
  type string;
  description "name of rule";
}
leaf pkts-matched {
  type uint64;
  description "number of
    packets that matched filter";
}
leaf pkts-modified {
  type uint64;
  description "number of
    packets that filter caused
    to be modified";
}
    leaf pkts-dropped {
      type uint64;
      description "number of
        packets that filter caused
        to be modified";
    }
leaf bytes-dropped {
  type uint64;
  description "number of
    packets that filter caused
    to be modified";
}
leaf pkts-forwarded {
  type uint64;
  description "number of
    packets that filter caused
    to be forwarded.";
}
leaf bytes-forwarded {
  type uint64;
  description "number of
    packets that filter caused
    to be forwarded.";
}

description "list of
operational statistics for each
rule.";
```



```
    }
    description "statistics
        on packet filter matches, and
        based on matches on many were
        modified and/or forwarded";
}

grouping packet-size-match {
    leaf l1-size-match {
        type uint32;
        description "L1 packet match size.";
    }
    leaf l2-size-match {
        type uint32;
        description "L2 packet match size.";
    }
    leaf l3-size-match {
        type uint32;
        description "L3 packet match size.";
    }
    leaf l4-size-match {
        type uint32;
        description "L4 packet match size.";
    }
    leaf service-meta-size {
        type uint32;
        description "service meta info match size.";
    }
    leaf service-meta-payload {
        type uint32;
        description "service meta-play match size";
    }
    description "packet size by layer
        only non-zero values are matched";
}

grouping time-day-match {

    description "matches for
        time of day.";
}

grouping eca-matches {
    uses interface-match;
    uses L1-header-match;
    uses L2-header-match;
```

```
    uses L3-header-match;
    uses L4-header-match;
    uses service-header-match;
    uses packet-size-match;
    uses time-day-match;
    description "ECA matches";
}

grouping eca-qos-actions {
  leaf cnt-actions {
    type uint32;
    description "count of ECA actions";
  }
  uses interface-actions;
  uses L1-header-actions;
  uses l2-header-mod-actions;
  uses L3-header-actions;
  uses L4-header-actions;

  description "ECA set or change
    packet Actions. Actions may be
    added here for interface,
    L1, L2, L3, L4 nad service forwarding
    headers.";
}

grouping ip-next-fwd {
  leaf rib-name {
    type string;
    description "name of RIB";
  }
  leaf next-hop-name {
    type string;
    description "name of next hop";
  }
  description "ECA set or change
    packet Actions";
}

grouping eca-fwd-actions {
leaf interface-fwd {
  type if:interface-ref;
  description "name of interface to forward on";
}
uses i2rs-rib:nexthop;
uses ip-next-fwd;
leaf drop-packet {
  type boolean;
}
```

```
        description "drop packet flag";
    }
    description "ECA forwarding actions";
}

grouping pkt-eca-policy-set {
    list groups {
        key "group-name";
        leaf group-name {
            type string;
            description
                "name of group of rules";
        }
        leaf vrf-name {
            type string;
            description "VRF name";
        }
        uses rt:address-family;
        list group-rule-list {
            key "rule-name";
            leaf rule-name {
                type string;
                description "name of rule";
            }
            leaf rule-order-id {
                type uint16;
                description "rule-order-id";
            }
        }
        description "rules per group";
    }
    description "pkt eca rule groups";
}
list eca-rules {
    key "order-id eca-rule-name";
    ordered-by user;
    leaf order-id {
        type uint16;
        description "Number of order
            in ordered list (ascending)";
    }
    leaf eca-rule-name {
        type string;
        description "name of rule";
    }
    leaf installer {
        type string;
        description

```

```
        "Id of I2RS client
        that installs this rule.";
    }
    uses eca-matches;
    uses eca-qos-actions;
    uses eca-fwd-actions;
    description "ECA rules";
  } // end of rule
description "Policy sets.";
}

grouping pkt-eca-opstate {
  uses groups-status;
  uses rule-group-link;
  uses rules_opstate;
  uses rules_opstats;
  description "pkt eca policy
  op-state main";
}

container pkt-eca-policy-opstate {
  config "false";
  uses pkt-eca-opstate;
  description "operational state";
}
}
<CODE ENDS>
```

6. IANA Considerations

This draft requests IANA Assign a urn in the IETF yang module space for:

```
"urn:ietf:params:xml:ns:yang:ietf-pkt-eca-policy";
```

```
associated prefix "pkt-eca";
```

7. Security Considerations

These generic filters are used in the I2RS FB-RIBs to filter packets in a traffic stream, act to modify packets, and forward data packets. These I2RS filters operate dynamically at same level as currently deployed configured filter-based RIBs to filter, change, and forward traffic. The dynamic nature of this protocol requires that I2RS Filters track the installer of group information and rules.

This section will be augmented after a discussion with security experts.

8. Informative References

- [I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-12 (work in progress), December 2015.
- [I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-08 (work in progress), October 2015.
- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-09 (work in progress), December 2015.
- [I-D.ietf-netmod-acl-model]
Bogdanovic, D., Koushik, K., Huang, L., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-06 (work in progress), December 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3060] Moore, B., Ellessen, E., Strassner, J., and A. Westerinen, "Policy Core Information Model -- Version 1 Specification", RFC 3060, DOI 10.17487/RFC3060, February 2001, <<http://www.rfc-editor.org/info/rfc3060>>.
- [RFC3460] Moore, B., Ed., "Policy Core Information Model (PCIM) Extensions", RFC 3460, DOI 10.17487/RFC3460, January 2003, <<http://www.rfc-editor.org/info/rfc3460>>.
- [RFC3644] Snir, Y., Ramberg, Y., Strassner, J., Cohen, R., and B. Moore, "Policy Quality of Service (QoS) Information Model", RFC 3644, DOI 10.17487/RFC3644, November 2003, <<http://www.rfc-editor.org/info/rfc3644>>.

Authors' Addresses

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Russ White
Ericsson

Email: russw@riw.us

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

S. Hares
Huawei
A. Beirman
YumaWorks
A. Dass
Ericsson
March 21, 2016

I2RS protocol strawman
draft-hares-i2rs-protocol-strawman-01.txt

Abstract

This document provides a strawman proposal for the I2RS protocol covering the ephemeral data store and data flow requirements not covered by I2RS publication/subscription service or traceability. It also proposes additions to YANG for the ephemeral data store and for additional data flow requirements. It proposes additions to the NETCONF and RESTCONF for these additions. Future versions of this document will propose changes to support the 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 <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 September 22, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

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

Table of Contents

1. Introduction	3
1.1. Ephemeral Changes	4
1.2. Data Flow Changes	4
2. Definitions Related to Ephemeral Configuration	5
3. Definition of ephemeral datastore for NETCONF/RESTCONF	6
4. Error handling	9
4.1. Error handling: I2RS Normal handling	9
4.2. Error Handling: Multiple I2RS Clients Write Same Node	10
4.3. Error handling: Basic Impact on functions	10
4.3.1. Initial Support of Parital Writes	10
4.3.2. Future Scope of multiple messagewrites	10
4.3.3. Grouping and Error handling	11
4.4. Error Handling: Different levels of Validation (Debate topic)	11
4.4.1. Validation during security outage	12
4.4.2. Solution ideas	12
4.4.3. Impact on NETCONF/RESTCONF functions	13
5. transport protocol	15
5.1. Secure Protocols	15
5.2. Insecure Protocol	15
6. Yang Library Use by Ephemeral	16
7. Simple Thermostat Model	17
7.1. Yang changes	19
7.2. RESTCONF sequence	20
7.3. NETCONF messages	20
8. NETCONF protocol extensions for the ephemeral datastore	21
8.1. Overview	21
8.2. Dependencies	22
8.3. Capability identifier	22
8.4. New Operations	22
8.4.1. Bulk-Write	22
8.5. Modification to existing operations	23
8.5.1. <get-config>	23
8.5.2. <edit-config>	23
8.5.3. <copy-config>	24
8.5.4. <delete-config>	25
8.5.5. <lock> and <unlock>	25
8.5.6. <get>	25
8.5.7. <close-session> and <kill-session>	25

8.6.	Interactions with Capabilities	25
8.6.1.	writable-running and candidate datastore	25
8.6.2.	confirmed commit	26
8.6.3.	rollback-on-error	26
8.6.4.	validate	26
8.6.5.	Distinct Startup Capability	26
8.6.6.	URL capability and XPATH capability	27
9.	RESTCONF protocol extensions for the ephemeral datastore . .	27
9.1.	Overview	27
9.2.	Dependencies	27
9.3.	Capability identifier	27
9.4.	New Operations	27
9.5.	modification to data resources	28
9.6.	Modification to existing operations	28
9.6.1.	OPTIONS changes	28
9.6.2.	HEAD changes	28
9.6.3.	GET changes	28
9.6.4.	POST changes	28
9.6.5.	PUT changes	29
9.6.6.	PATCH changes	29
9.6.7.	DELETE changes	29
9.6.8.	Query Parameters	29
9.7.	Interactions with Notifications	29
9.8.	Interactions with Error Reporting	29
10.	IANA Considerations	29
11.	Security Considerations	30
12.	Acknowledgements	30
13.	Major Contributors	30
14.	References	31
14.1.	Normative References:	31
14.2.	Informative References	33
	Authors' Addresses	33

1. Introduction

This documents is a strawman for I2RS higher level protocol. The I2RS protocol is a higher level protocol comprised of a set existing protocols which have been extended to work together to support a new interface to the routing system. Some people are suggesting only two protocols should be defined: NETCONF [RFC6241], and RESTCONF [I-D.ietf-netconf-restconf]. Others are suggesting we should include other data protocols.

This draft is input to a NETCONF review and design team. Many items have been settled on. Some items are in debate and those titles of those sections are marked.

This strawman proposal for the I2RS protocol covers the ephemeral data store and data flow requirements not covered by I2RS publication/subscription service or traceability. It also proposes additions to YANG for the ephemeral data store and for these additional data flow requirements. It also proposes extensions to NETCONF and RESTCONF to support ephemeral state and I2RS.

draft-hares-i2rs-protocol-strawman-examples (pending) provides examples of this strawman protocol use for I2RS. This draft uses a simple thermostat model to illustrate commands.

1.1. Ephemeral Changes

This document proposes additions to support ephemeral state in the datastores supported by NETCONF and RESTCONF, and in the YANG modules that define these data stores. The requirements for the I2RS ephemeral state are covered in [I-D.ietf-i2rs-ephemeral-state]

This draft provides suggests the following additions to support the I2RS ephemeral state:

- o Yang ephemeral statement,
- o NETCONF ([RFC6241]) protocol extensions for the ephemeral data store,
- o RESTCONF ([I-D.ietf-netconf-restconf]) protocol extensions for the ephemeral data store

1.2. Data Flow Changes

This document proposes additions to support data flows from different data models for large data flows, traffic monitoring, actions and OAM interaction, and flows during outages or attacks. The requirements for these changes are define in [I-D.hares-i2rs-dataflow-req].

Most large data flows will be handled utilizing the publication/subscription service define in the I2RS publication/subscription service requirements specified in [I-D.ietf-i2rs-pub-sub-requirements]. Extensions to NETCONF to support a push publication/subscription service have been defined in [I-D.ietf-netconf-yang-push]. This document does not propose a pull publication/subscription (pull pub-sub) service for the first set of component protocols for the I2RS higher level protocol. If deployments require the pull pub-sub service, then an expansion of the push service can provide one mechanism.

This document does provide support for the I2RS protocol to:

Support large data transfers in a data agnostic format (DF-REQ-02) supporting transfers of data in any format (E.g. XML, JSON, MTL, protobuf, ASCII) over any transport (DF-REQ-03).

Support the use of IPFIX as a component protocol to send traffic monitoring data or any type of large data flow from I2RS agent to I2RS client (DF-REQ-04),

Support exporting traffic statistics for filter-based policy usage (BGP-FS, I2RS FB-FIB, policy routing), IPPM, SFLOW and other traffic statistics using either yang models or IPFIX template formats over any data encapsulation format over any transport (DF-REQ-05).

2. Definitions Related to Ephemeral Configuration

Currently the configuration systems managed by NETCONF ([RFC6241]) or RESTCONF ([I-D.ietf-netconf-restconf]) have three types of configuration: candidate, running, and startup running under the config=true flag.

- o The candidate receives configuration changes from NETCONF/RESTCONF.
- o The running configuration is the configuration currently operating on a devices
- o The start-up configuration is the configuration that survives a reboot.

The config=false flag has operational data which exists alongside the config=true data. However, at this point there is no data stored defined for configuration false.

```

.....      .....      .....
:Candidate : --> : running : --> :start-up  :
.....      .....      .....

config true
-----
config false
```

Figure 1

The [I-D.ietf-netmod-opstate-reqs] defines new terms to clarify how this works. In reality, the running configuration becomes the intended configuration that is intended to be loaded into a device.

The loading of the update into the system can be either asynchronous or synchronous. In the asynchronous case, the NETCONF server responds to the client after the intended has been updated, but the applied configuration is only updated later when the configuration change has full impacted all components on the device. The synchronous configuration operation occurs when both the intent configuration has been updated and the actual configuration has been loaded after resolving the necessary things to load in a box.

This document will use the terms defined in [I-D.ietf-netmod-opstate-regs].

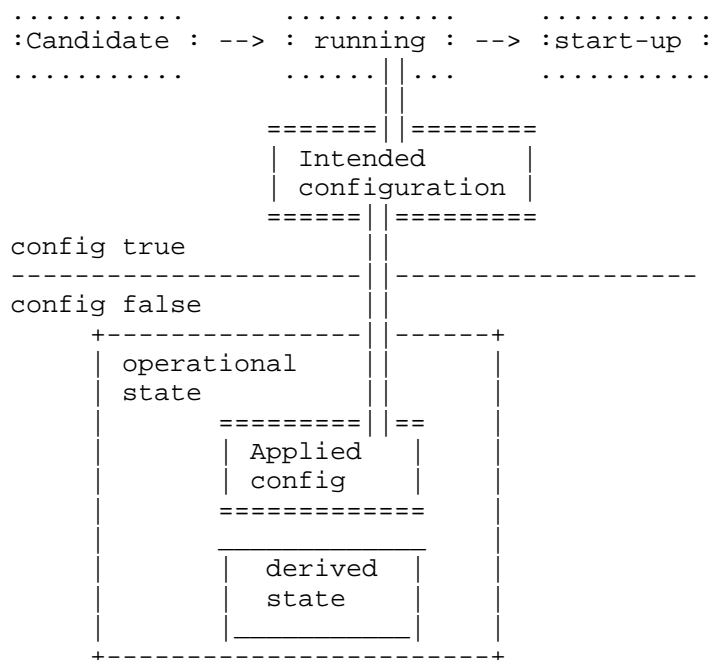


Figure 2

3. Definition of ephemeral datastore for NETCONF/RESTCONF

This section describes the properties of the ephemeral datastore. The ephemeral datastore is not unique to I2RS. This approach to the ephemeral datastore is a panes-of-glass model. This definition of I2RS does not support caching in the I2RS Agents. Future I2RS work may reconsider supporting caching.

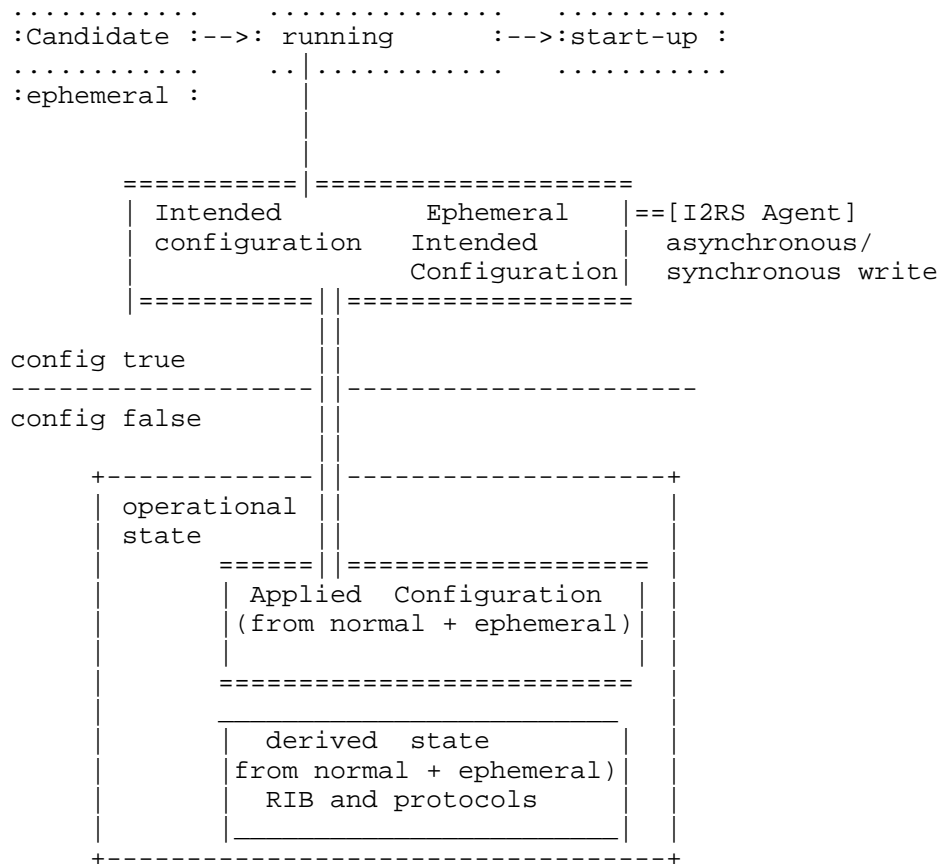


Figure 3

The ephemeral data store has the following qualities:

1. Ephemeral state is not unique to I2RS work.
 2. The ephemeral datastore is never locked.
 3. The ephemeral datastore is really a portion of the intended configuration that does not persist over a reboot.
- * Since Ephemeral is just about data not persisting over a reboot, then in theory any node or group of nodes in a YANG data model could be ephemeral. The YANG data module must indicate what portion of the data model (if any) is ephemeral.

- * A YANG data module could be all ephemeral (e.g. [I-D.ietf-i2rs-rib-data-model]) with no directly associated configuration models,
 - * A YANG model could be all ephemeral but associated with a configuration model (E.g. [I-D.hares-i2rs-bgp-dm],
 - * or a single data node or data tree could be made ephemeral.
4. The applied configuration is the result of the the intent configuration (normal and ephemeral). Similarly, the derived data is a result of the applied configuration.
 5. Ephemeral portions (node, tree, or data model) need to be signalled in the conformance portions of the NETCONF and RESTCONF work. Conformance is signalled in the following ways:
 - * The conformance portion of NETCONF ([RFC6241]) is currently signalled in the <hello>.
 - * Yang 1.1 and RESTCONF uses the module library ([I-D.ietf-netconf-yang-library])
 - * NETCONF may use the module library in the future.
 - * The ephemeral status in a module will be listed as "all, none or partial". Optionally the module may provide a list of nodes.
 6. The ephemeral data store is treated as one pane of glass that an I2RS client(s) may read/write which has the following implications:
 - * The ephemeral datastore overlays the configuration datastore at the intended configuration. By overlays, the I2RS write overwrites a previous configuration value, but if a local configuration value changes after that over-write the default is to have the local-config win. [aka Last Write wins.]
 - + An example may help to illustrate this default rule. Say a configuration specifies a local route of 128.2/16 with a nexthop of 192.5.10.1. Afterwards an ephemeral route is added for 128.2/16 with nexthop of 192.5.10.2. This ephemeral route would replace the first route. If the configuration changes the underlying route (128.2/16 with nexthop of 192.5.10.1) and the default rule of local configuration is in effect, the local configuration value (128.2/16 with nexthop of 192.5.10.1) would take effect.

This follows the normal netconf concept that Last configured wins. The I2RS agent would notify the I2RS Client that the ephemeral route (128.2/16 with nexthop of 192.5.10.2) had been overwritten by the local configuration.

- * The default of local can be changed by operator-applied policy to allow ephemeral to always win or local configuration to always win, but the status of the operator applied policy must be queryable in the I2RS agent (if that scope) or in the I2RS ephemeral data model. I2RS clients are required to understand and handle if the an I2RS agent supports something different than the default (aka Last write wins).

4. Error handling

This section will go over I2RS normal error handling, error handling when multiple I2RS clients write to the same node, and suggested alterations to the validation process for nodes.

Editor's note: The requirement for alterations to validation needs to be confirmed.

4.1. Error handling: I2RS Normal handling

Normal error handling of I2RS Agent for an I2RS client's information examines the following:

- o message syntax validation,
- o syntax validation for nodes of data model,
- o removes referential requirements for leafref checking, MUST clauses, and instance identifier,
- o grouping of data within a data model or across data models to accomplish tasks,
- o permission to write nodes of data model,
- o grouping,
- o priority to write nodes of data model being higher than existing priority

The full error handling status includes all checks included for any normal YANG data module used by NETCONF/RESTCONF. This includes

referential checks for leafref checks, MUST clauses, and instance identifiers.

If the I2RS protocol allows agents to set permissible range of error handling for writes on a data model (none, I2RS normal, full), then those stating this requirement want to be able to change this with operator-applied settings (e.g. always request full validation).

4.2. Error Handling: Multiple I2RS Clients Write Same Node

Multiple I2RS clients writing to the same variable is considered an "error condition" in the I2RS architecture [I-D.ietf-i2rs-architecture], but the I2RS Agent must handle this error condition. Upon multiple I2RS clients writing, the ephemeral data store allows for priority pre-emption of the write operation. Priority pre-emption means each I2RS client of the ephemeral I2RS agent (netconf server) is associated with a priority. Priority pre-emption occurs when a I2RS client with a higher priority writes a node which has been written by an I2RS client (with the lower priority). At this point, the I2RS agent (netconf server) allows the write and provides a notification indication to the notification publication/subscription service.

4.3. Error handling: Basic Impact on functions

4.3.1. Initial Support of Parital Writes

The initial releases of I2RS will only require "all-or-nothing" in the I2RS Agent.

4.3.1.1. NETCONF Support of Partial Writes

NETCONF does not support a mandated sequencing of edit functions or write functions. Without this mandated sequences, NETCONF cannot support partial edits.

4.3.1.2. RESTCONF Support of Partial Writes

RESTCONF has a complete set of operations per message. The RESTCONF patch can support write functions per messages.

4.3.2. Future Scope of multiple message writes

Error handling on writes of the ephemeral datastore is different for nodes that are grouped versus orthogonal. Group nodes may need to be all changed or all removed (all-or-nothing). In contrast, writing orthogonal data nodes in the same data module or between data models need to be added or deleted in sync.

The [I-D.ietf-i2rs-architecture] specifies three types of error handling for a partial write operation: "all-or-nothing", "stop-on-error", or "continue-on-error". Partial write operations of "stop-on-error" or "continue-on-error" are allowed only for data writes which are not a part of a grouping within a data model. The definition of the I2RS error conditions are:

- o stop-on-error - means that the configuration process stops when a write to the configuration detects an error due to write conflict.
- o continue-on-error - means the configuration process continues when a write to the configuration detects an error due to write process, and error reports are transmitted back to the client writing the error.
- o all-or-nothing - means that all of the configuration process is correctly applied or no configuration process is applied. (Inherent in all-or-nothing is the concept of checking all changes before applying.)

4.3.3. Grouping and Error handling

Yang 1.0 and Yang 1.1 provide the ability to group data in groupings, leafref lists, lists, and containers. Grouping of data within a model links to data that is logically associated with one another. Data models may logical group data across models. One example of such an association is the association of a static route with an interface. The concepts of groupings apply to both ephemeral and non-ephemeral nodes within a data model.

4.4. Error Handling: Different levels of Validation (Debate topic)

The requirement for Ephemeral nodes level of validation/error handling in the I2RS protocol have been suggested to have three types of validation based on an operator-applied policy for I2RS protocol.

- o syntax validation only,
- o Ephemeral data store allows for reduced error handling that removes the requirements for referential checks [I2RS normal error handling]
- o ephemeral data store handling that uses normal NETCONF/RESTCONF error handling with syntax and referential [full],

Editor's note: Andy Bierman believes that only full-validation will work. Kent Watsen suggested the "no-referential checks". Jan Medved suggested the "syntax only checks". Three excellent engineers who

are implementing I2RS suggested these three features. The editor needs aid to discuss the details of this requirement and proposal.

The first step is to see if we can confirm the requirement. After we've confirmed the requirement, the second step is to have a detailed discussion about the pro/cons of this validation. We expect to do this at IETF95.

4.4.1. Validation during security outage

[I-D.hares-i2rs-dataflow-req] indicates that higher levels of validity need to occur during security attacks. Network security controllers communicate with routing devices with network security functions such as basic firewalls in order change firewall settings during attacks. The I2NSF WG is defining communication between the network security controllers and the NSF/vNSF functions in the routers and other network devices. [I-D.hares-i2nsf-mgtflow-reqs] describes the challenges to management information flow between NSF controllers and NSF/vNSF devices operating correctly or effectively during DDoS or network security attacks.

Higher referential checks may be useful during these periods of security attacks (DDoS or others).

4.4.2. Solution ideas

This section is written to provide ideas for that discussion.

If the I2RS protocol is required to have three levels of error handling (syntax only, no-referential, full), the following are ideas for solutions:

1. only allow full validation,
2. allow a particular set of validation (syntax checks, no-referential, all-checks) per deployments of an I2RS Agent (operator-applied selection of error checking on the whole system),
3. Restrict the use of the "syntax only to operator-applied error checking" (argument: if the operator wants to shoot himself in the foot, fine). Note any module, submodule, or node that has this feature.
4. Restrict the the use of "no-referential checking to I2RS independent protocol modules, and provide error reports of referential checks,

4.4.3. Impact on NETCONF/RESTCONF functions

This section describes the ephemeral data stores handling for each of the functions.

4.4.3.1. Syntax validation

Syntax validation of the message should be done according to the NETCONF or RESTCONF protocol features. New features for ephemeral datastore should provide the error handling with the feature. Message syntax validation can be for read, write, or rpc functions.

Syntax validation of the data model included in the ephemeral data store should be done by I2RS Agent.

4.4.3.2. Referential validation

The ephemeral data store normal processing does not do the following referential checks: leafref, MUST, instance identifier. The removal of these validations allows for intelligent I2RS clients to rapidly read or write data, and handle error conditions at a higher level.

4.4.3.3. Grouping and Error handling

Yang 1.0 and Yang 1.1 provide the ability to group data in groupings, leafref lists, lists, and containers. Adding the ephemeral data store will add these rules to references between data stores:

1. Ephemeral node can refer to config nodes, or derived state nodes (e.g. LSP),
2. config nodes cannot refer to ephemeral intended configuration nodes, and
3. derived state nodes can refer to ephemeral configuration or configuratino nodes.
4. derived state nodes are "non-persistent" and may disappear if a protocol event occurs
5. ephemeral datastore nodes are "non-presistent" and will disappear upon a reboot of the software/hardware.

Referential checks require the above rules. Not doing referential checks could cause one or more broken references to exist in the ephemeral data base. An ephemeral data bases with broken references may crash, given faulty information, or perform wrong protocol actions.

4.4.3.4. Priority preemption

I2RS protocol uses priority to resolve two I2RS clients having permissions to write the same pieces of data in an I2RS agent (NETCONF server). If two (or more) I2RS clients attempt to write the same data, the one with the highest priority is enabled to write the data. In the case of two clients with the same priority attempting to write data, the first one to request write wins.

Each client has a unique priority. Client identities and priorities are assigned outside of I2RS by exterior mechanisms such as AAA or administrative interfaces. A valid I2RS client must have both an identity and a priority.

A client-id and priority must be saved per node.

A sample container for I2RS client information is shown below.

```
container i2rs-clients {
  leaf max-clients {
    config false;
    mandatory true;
    type uint32 {
      range "1 .. max";
    }
  }
  list i2rs-client {
    key name;
    unique priority;
    leaf name { ... }
    leaf priority { ... }
  }
}
```

Figure 4

4.4.3.4.1. Andy Bierman Priority Comment

(Andy) This priority is not required to be densely numbered. Whether there are 1 pane per client or 1 pane per priority or 1 giant blob full of everything, the code will be the same. The goal of "unique priority" is to require that only priority be saved in the meta-data for the ephemeral datastore. Without that, client-id and priority must be saved (per data node).

5. transport protocol

5.1. Secure Protocols

NETCONF's XML-based protocol ([RFC6241]) can operate over the following secure and encrypted transport layer protocols:

SSH as defined in [RFC6242],

TLS with X.509 authentication [RFC7589]

RESTCONF's XML-based or JSON [RFC7158] data encodings of Yang functions are passed over HTTOS with (GET, POST, PUT, PATCH, DELETE, OPTIONS, and HEAD).

5.2. Insecure Protocol

The ephemeral database may support insecure protocols for information which is ephemeral state which does not engage in configuration. The insecure protocol must be defined in conjunction with a data model or a subdata model.

[RFC6536] has two extensions for security. Two extensions supporting ephemeral and insecure might look like:

```
extension ephemeral {  
  description "if present in a data definition statement  
    then the object is considered OK for editing as ephemeral data."  
}  
extension non-secure-ok {  
  description "if present in data definition statement  
    then the object is considered OK for non-secure transport."}
```

```
T declare a local config and ephemeral edit:
leaf both {
  i2rs:ephemeral;
  type string;
  config true;
  // Yang allows leafref/XPATH to point at config=true only
}
```

```
To declare an object ephemeral edit only
leaf eph {
  i2rs:ephemeral;
  type string;
  config false;
}
```

```
To declare a non-secure leaf
leaf in-octets {
  i2rs:nonsecure-ok;
  type yang:counter64;
  config false;
}
```

6. Yang Library Use by Ephemeral

The data modules supporting the ephemeral datastore can use the Yang module library to describe their datastore. Figure 5 shows the module library data structure as found [I-D.ietf-netconf-yang-library].

The I2RS modules will provide features for I2RS ephemeral state and protocol of:

- o protocol version support - "version 1",
- o ephemeral model scope - ephemeral modules, mixed config module (ephemeral and config), mixed derived state (ephemeral and config).
- o multiple message support - "all or nothing",
- o pane of glass support - "single only".
- o protocol supported - "NETCONF", "RESTCONF", "NETCONF pub-sub push",
- o encoding support - XML or JSON

```

o transports protocol supported: "TCP", "SSH", "TLS", non-secure,
  and others.

o configuration for non-secure transport (An example is

  * i2rs:nonsecure-ok;

  )

+--ro modules
  +--ro module*[name revision]
    +--ro name yang:yang-identifier
    +--ro revision union;
    +--ro schema? inet:uri
    +--ro namespace inet:uri
    +--ro feature* yang:yang-identifier
    +--ro deviation* [name revision]
      | +--ro name yang:yang-identifier
      | +--ro revision union
    +--ro conformance enumeration
    +--ro submodules
      +--ro submodule*[name revision]
        +--ro name yang:yang-identifier
        +--ro revision union
        +--ro schema? inet:uri

```

Figure 5

Editor's Note: One feature under debate is data modules providing different levels of check on rpc or writes.

ephemeral checking - syntax only, no-referential, and full checking.

7. Simple Thermostat Model

In this discussion of ephemeral configuration, this draft utilizes a simple thermostat model with the YANG configuration found in figure 6.

```
module thermostat {  
  ..  
  leaf desired-temp {  
    type int32;  
    units "degrees Celsius";  
    description "The desired temperature";  
  }  
  
  leaf actual-temp {  
    type int32;  
    config false;  
    units "degrees Celsius";  
    description "The measured temperature  
    (operational state).";  
  }  
}
```

Figure 6 - Simple thermostat YANG Model

Figure 6 shows two I2RS clients talking to this model: scheduler and hold-temp. Scheduler has a schedule set of temperatures to put in the thermostat. Hold-temp holds the temperature at the same value. The hold-temp I2RS client has a higher priority than the scheduler client.

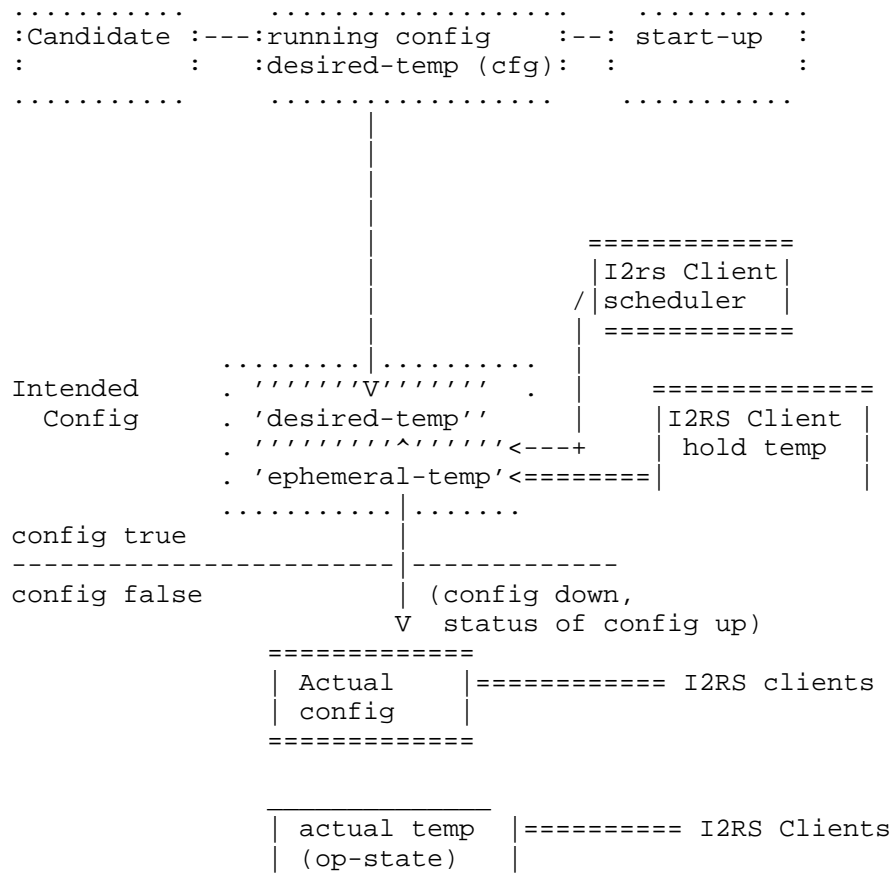


Figure 6 - Two I2RS clients

7.1. Yang changes

```
module thermostat {  
    ..  
  
    leaf desired-temp {  
        type int32;  
        units "degrees Celsius";  
        ephemeral true;  
        description "The desired temperature";  
    }  
  
    leaf actual-temp {  
        type int32;  
        config false;  
        units "degrees Celsius";  
        description "The measured temperature";  
    }  
}
```

Figure 7 - Simple Thermostat Yang with ephemeral

7.2. RESTCONF sequence

Figure 7 shows the thermostat model has ephemeral variable desired-temp in the running configuration and the ephemeral data store. The RESTCONF way of addressing is below:

RESTCONF running data store

```
PUT /restconf/data/thermostat:desired-temp  
{ "desired-temp":18 }
```

RESTCONF ephemeral datastore

```
PUT /restconf/data/thermostat:desired-temp?datastore=ephemeral  
{ "desired-temp":19 }
```

Figure 8 - RESTCONF setting of ephemeral state

7.3. NETCONF messages

The NETCONF way of transmitting this data would be

```
<rpc-message-id=101
  xmlns="urn:ietf:params:xml:ns:base:1.0">
  <edit-config>
    <target>
      <ephemeral >
        true
      </ephemeral >
    </target>
    <config>
      <top xmlns="http://example.com/schema/1.0/thermostat/config">
        <desired-temp> 18 </desired-temp>
      </top>
    </config>
  </edit-config>
</rpc>
```

Note: config=TRUE; datastore = ephemeral
ephemeral-validation=full-check;

figure 8 NETCONF setting of desired-temp

8. NETCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-datastore

8.1. Overview

This capability defines the NETCONF protocol extensions for the ephemeral state. The ephemeral state has the following features:

- o the ephemeral datastore is a datastore that holds configuration information (Config=true) that is intended to not survive a reboot.
- o The ephemeral capability is signalled as a capability for a node, a sub-module, or a module either in the conformance portion of NETCONF (<hello>) or via netconf yang module library ([I-D.ietf-netconf-yang-library]) used by Yang 1.1 and RESTCONF.
- o ephemeral data will be noted by an "ephemeral statement at the node or module "
- o The ephemeral datastore is never locked.
- o The ephemeral data store is one pane of glass that overrides the intended config which is normally the running datastore, but can be designated as the candidate config.

- o Ephemeral data nodes can occur as part of protocol or protocol independent modules. However, ephemeral data nodes cannot have non-ephemeral data nodes within the subtree. Ephemeral sub-modules cannot have non-ephemeral data nodes within the module. Ephemeral modules cannot have non-ephemeral sub-modules or nodes within the module.
- o ephemeral writes have two checks: error validation and priority preemption between two I2RS client writes to the same data.
- o ephemeral error checking has the following three levels
 - * syntax only - message and data module syntax,
 - * reduced error checking that remove the requirement for leafref checking, MUST clauses, and instance identifier validation.

The default is reduced error checking.

- o The write operation with a priority pre-emption by a higher priority client of the lower priority clients write where the overwrite triggers a notification by the I2RS agent to the lower priority client.

8.2. Dependencies

The following are the dependencies for ephemeral support:

The Yang data modules must be flag with the ephemeral data store at the node, sub-module and model.

(under debate) Yang data models must specify ephemeral validation if the models desire validation other reduced error checking.

The Yang modules must support the notification of write-conflicts.

8.3. Capability identifier

The ephemeral-datastore capability is identified by the following capability string: (capability uri)

8.4. New Operations

8.4.1. Bulk-Write

Bulk Write allows for large scale writes with error handling that is specified as syntax or reduced or full. Alternatively, the data modules can utilize an RPC to do bulk reads/writes. The bulk write

will be first check for other I2RS clients having a higher priority write value for any of the values.

Editor: Do we need something beyond rpc for bulk data writes?

8.5. Modification to existing operations

The capability for :ephemeral-datastore modifies the target for existing operations.

8.5.1. <get-config>

The :ephemeral-datastore capability modifies the <edit-config> to accept the <ephemeral> as a target for source, and allows the filters focused on a particular module, submodule, or node.

The positive and negative responses remain the same.

Example - retrieve users subtree from
ephemeral database

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <ephemeral-datastore/>
    </source>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.0/thermostat/config">
        <desired-temp>
        </top>
      </filter>
    </get-config>
  </rpc>
```

8.5.2. <edit-config>

The :ephemeral-datastore capability modifies the <edit-config> to accept the <ephemeral> as a target for source with filters. The operations of merge, replace, create, delete, and remove are available, but each of these operations is modified by the priority write as follows:

<merge> parameter is replaced by <merge-priority> The current data is modified by the new data in a merge fashion only if existing data either does not exist, or is owned by a lower priority client. If any data is replaced, this event is passed to the notification function within the pub/sub and traceability.

<replace> is replaced by <replace-priority> for ephemeral datastore which replaces data if the existing data is owned by a lower priority client. If data any data is replaced, this event is passed to the notification function within pub/sub and traceability for notification to the previous client. The success or failure of the event is passed to traceability.

<create> - the creation of the data node works as in [RFC6241] except that the success or failure is passed to pub/sub and traceability functions.

<deletion> - the deletion of the data node works as in [RFC6241] except event that the success or the error event is passed to the notification services in the pub/sub and traceability functions.

<remove> - the remove of the data node works as in [RFC6241] except that all results are forwarded to traceability.

The existing parameters are modified as follows:

<target> - add a target of :ephemeral-datastore

<default-operation> -allows only <merge-priority> or <replace-priority>

<error-option> - the I2RS agent agent supports only the a "all-or-nothing" equivalent to a "rollback-on-error" function.

positive response - the <ok> is sent for a positive response within an <rpc-reply>.

negative response - the <rpc-error> is sent for a negative response within an <rpc-reply>. Note a negative responses may evoke a publication of an event.

8.5.3. <copy-config>

Copy config allows for the complete replacement of all the ephemeral nodes within a target. The alternation is that source is the :ephemeral datastore with the filtering to match the datastore. The following existing parameters are modified as follows:

<target> - add a target of :ephemeral-datastore

<error-option> - the I2RS agent agent supports only the a "all-or-nothing" equivalent to a "rollback-on-error" function.

positive response - the <ok> is sent for a positive response within an <rpc-reply>.

negative response - the <rpc-error> is sent for a negative response within an <rpc-reply>.

8.5.4. <delete-config>

The delete will delete all ephemeral nodes out of a datastore. The target parameter must be changed to allow :ephemeral-datastore. and filters.

8.5.5. <lock> and <unlock>

Lock and unlock are not supported with a target of :ephemeral-datastore.

8.5.6. <get>

The <get> is altered to allow a target of :ephemeral-datastore and with the filters.

8.5.7. <close-session> and <kill-session>

The close session is modified to take a target of :ephemeral-datastore, Since no locks are set, none should be released.

The kill session is modified to take a target of "ephemeral-datastore. Since no locks are set, none should be released.

8.6. Interactions with Capabilities

[RFC6241] defines NETCONF capabilities for writeable-running datastore, candidate config data store, confirmed commit, rollback-on-error, validate, distinct start-up, URL capability, and XPATH capability. I2RS ephemeral state does not impact the writeable-running data store or the candidate config datastore.

8.6.1. writable-running and candidate datastore

The writeable-running and the candidate datastore cannot be used in conjunction with the ephemeral data store. Ephemeral database overlays an intended configuration, and does not impact the writable-running or candidate data store.

8.6.2. confirmed commit

Confirmed commit capability is not supported for the ephemeral datastore.

8.6.3. rollback-on-error

The rollback-on-error when included with ephemeral state allows the error handling to be "all-or-nothing" (rollback-on-error).

8.6.4. validate

Editorial: Andy Bierman feels that any validation except full is going to leave the ephemeral datastore unusable. Kent Watsen suggested a "no-referential" validation as the default for I2RS protocol. Jan Medved indicated that many of the ODL Route updates are validated on the I2RS client extensively, so that the update can occur quickly with a "syntax only". Three operations people have indicated 3 different implementations. This needs to be discussed at IETF.

The text below is only a command that would provide a key word to allow three different types of validation. The command gives form to the requirements and comments from others, but it may also be broken.

The <validate> key word is expanded to support the following:

source: ephemeral-datastore

validate: (syntax, no-referential, full) with the following definitions:

- * syntax - validates only the message syntax and the data base syntax.
- * no-referential - skips referential test (leafref, MUST clauses, and instance identifiers).
- * full - all normal netconf/restconf module error checking

8.6.5. Distinct Startup Capability

This NETCONF capability appears to operate to load write-able running config, running-config, or candidate datastore. The ephemeral state does not change the environment based on this command.

8.6.6. URL capability and XPATH capability

The URL capabilities specify a <url> in the <source> and <target>. The initial suggestion to allow both of these features to work with ephemeral datastore.

9. RESTCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-datastore

9.1. Overview

This capability defines the RESTCONF protocol extensions for the ephemeral state. The ephemeral state has the features described in the previous section on NETCONF.

9.2. Dependencies

The ephemeral capabilities have the following dependencies:

Yang data nodes, sub-modules, or modules must be flagged with the config datastore flag;

The Yang modules must support the notification of write-conflicts.

The I2RS Yang modules must support the following:

the yang-patch features as specified in [I-D.ietf-netconf-yang-patch].

The yang module library feature [I-D.ietf-netconf-yang-library],

the equivalent of the netconf pub/subscription push service found in [I-D.ietf-netconf-yang-push]

9.3. Capability identifier

The ephemeral-datastore capability is identified by the following capability string: (capability uri)

9.4. New Operations

none

9.5. modification to data resources

RESTCONF must be able to support the ephemeral datastore as a context with its rules as part of the "{+restconf}/data" subtree. The "edit collision" features in RESTCONF must be able to provide notification to I2RS read functions or to rpc functions. The "timestamp" with a last modified features must support the traceability function.

The "Entity Tag" could support saving a client-priority tuple as a opaque string, but it is important that that additions be made to restore client-priority so it can be compared with strings can be done to determine the comparison of two I2RS client-priorities.

9.6. Modification to existing operations

The current operations in RESTCONF are: OPTIONS, HEAD, GET, POST, PUT, PATCH, and DELETE. This section describes the modification to these exiting operations.

9.6.1. OPTIONS changes

The options methods should be augmented by the [I-D.ietf-netconf-yang-library] information that will provide an indication of what ephemeral state exists in a data modules, or a data modules sub-modules or nodes.

9.6.2. HEAD changes

The HEAD in retrieving the headers of a resources. It would be useful to changes these headers to indicate the datastore a node or submodule or module is in (ephemeral or normal), and allow filtering on ephemeral nodes or trees, submodules or module.

9.6.3. GET changes

GET must be able to read from the URL and a context ("?context=ephemeral"). Similarly, it is important the Get be able to determine if the context=ephemeral.

9.6.4. POST changes

POST must simply be able to create resources in ephemeral datastores ("context=ephemeral") and invoke operations defined in ephemeral data models.

9.6.5. PUT changes

PUT must be able to reference an ephemeral module, sub-module, and nodes ("?context=ephemeral").

9.6.6. PATCH changes

Plain PATCH must be able to update or create child resources in an ephemeral context ("?context=ephemeral") The PATCH for the ephemeral state must be change to provide a merge or update of the original data only if the client's using the patch has a higher priority than an existing datastore's client, or if PATCH requests to create a new node, sub-module or module in the datastore.

9.6.7. DELETE changes

The phrase "?context=ephemeral" following an element will specify the ephemeral data store when deleting an entry.

9.6.8. Query Parameters

The query parameters (content, depth, fields, insert, point, start-time, stop-time, and with-defaults (report-all, trim, explicit, report-all-tagged) must support ephemeral context ("?context=ephemeral") described above.

9.7. Interactions with Notifications

The ephemeral database must support the ability to publish notifications as events and the I2RS clients being able to receiving notifications as Event stream. The event error stream processing should support the publication/subscription mechanisms for ephemeral state defined in [I-D.ietf-netconf-yang-push].

9.8. Interactions with Error Reporting

The ephemeral database must support in RESTCONF must also support passing error information regarding ephemeral data access over to RESTCONF equivalent of the and traceability client.

10. IANA Considerations

This is a protocol strawman - nothing is going to IANA.

11. Security Considerations

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

12. Acknowledgements

This document is an attempt to distill lengthy conversations on the I2RS proto design team from August

Here's the list of the I2RS protocol design team members

- o Alia Atlas
- o Ignas Bagdonas
- o Andy Bierman
- o Alex Clemm
- o Eric Voit
- o Kent Watsen
- o Jeff Haas
- o Keyur Patel
- o Hariharan Ananthakrishnan
- o Dean Bogdanavich
- o Anu Nair
- o Juergen Schoenwaelder
- o Kent Watsen

13. Major Contributors

- o Andy Bierman (Yuman Networks) - andy@yumaworks.com
- o Kent Watson (Juniper) (kwatsent@juniper.net)

14. References

14.1. Normative References:

- [I-D.hares-i2rs-dataflow-req]
Hares, S., "I2RS Data Flow Requirements", draft-hares-i2rs-dataflow-req-02 (work in progress), March 2016.
- [I-D.ietf-i2rs-architecture]
Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", draft-ietf-i2rs-architecture-13 (work in progress), February 2016.
- [I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-04 (work in progress), March 2016.
- [I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-03 (work in progress), March 2016.
- [I-D.ietf-i2rs-pub-sub-requirements]
Voit, E., Clemm, A., and A. Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-05 (work in progress), February 2016.
- [I-D.ietf-i2rs-rib-data-model]
Wang, L., Ananthakrishnan, H., Chen, M., amit.dass@ericsson.com, a., Kini, S., and N. Bahadur, "A YANG Data Model for Routing Information Base (RIB)", draft-ietf-i2rs-rib-data-model-05 (work in progress), March 2016.
- [I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-08 (work in progress), October 2015.
- [I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-00 (work in progress), October 2015.

- [I-D.ietf-i2rs-traceability]
Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", draft-ietf-i2rs-traceability-07 (work in progress), February 2016.
- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-10 (work in progress), March 2016.
- [I-D.ietf-netconf-yang-library]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library-04 (work in progress), February 2016.
- [I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-08 (work in progress), March 2016.
- [I-D.ietf-netconf-yang-push]
Clemm, A., Prieto, A., Voit, E., Tripathy, A., and E. Einar, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-01 (work in progress), February 2016.
- [I-D.ietf-netmod-opstate-reqs]
Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.
- [I-D.ietf-netmod-yang-metadata]
Lhotka, L., "Defining and Using Metadata with YANG", draft-ietf-netmod-yang-metadata-06 (work in progress), March 2016.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC7158] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7158, DOI 10.17487/RFC7158, March 2014, <<http://www.rfc-editor.org/info/rfc7158>>.

- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.

14.2. Informative References

- [I-D.hares-i2nsf-mgtflow-reqs]
Hares, S., "I2NSF Data Flow Requirements", draft-hares-i2nsf-mgtflow-reqs-00 (work in progress), March 2016.
- [I-D.hares-i2rs-bgp-dm]
Wang, L., Hares, S., and S. Zhuang, "An I2RS BGP Data Modell", draft-hares-i2rs-bgp-dm-00 (work in progress), October 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com

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

L. Wang
Individual
M. Chen
Huawei
A. Dass
Ericsson
H. Ananthakrishnan
Packet Design
S. Kini
Individual
N. Bahadur
Bracket Computing
May 14, 2018

A YANG Data Model for Routing Information Base (RIB)
draft-ietf-i2rs-rib-data-model-15

Abstract

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

Requirements Language

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

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 November 15, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<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	2
1.1. Definitions and Acronyms	3
1.2. Tree Diagrams	3
2. Model Structure	3
2.1. RIB Capability	7
2.2. Routing Instance and Rib	7
2.3. Route	8
2.4. Nexthop	9
2.5. RPC Operations	14
2.6. Notifications	18
3. YANG Modules	20
4. IANA Considerations	64
5. Security Considerations	65
6. Contributors	66
7. Acknowledgements	66
8. References	66
8.1. Normative References	66
8.2. Informative References	67
Authors' Addresses	68

1. Introduction

The Interface to the Routing System (I2RS) [RFC7921] provides read and write access to the information and state within the routing process that exists inside the routing elements, this is achieved via protocol message exchange between I2RS clients and I2RS agents associated with the routing system. One of the functions of I2RS is to read and write data of the Routing Information Base (RIB). [I-D.ietf-i2rs-usecase-reqs-summary] introduces a set of RIB use cases. The RIB information model is defined in [I-D.ietf-i2rs-rib-info-model].

This document defines a YANG [RFC7950][RFC6991] data model for the RIB that satisfies the RIB use cases and aligns with the RIB information model.

1.1. Definitions and Acronyms

RIB: Routing Information Base

FIB: Forwarding Information Base

RPC: Remote Procedure Call

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

1.2. Tree Diagrams

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

2. Model Structure

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

```

module: ietf-i2rs-rib
  +--rw routing-instance
    +--rw name string
    +--rw interface-list* [name]
    |   +--rw name if:interface-ref
    +--rw router-id? yang:dotted-quad
    +--rw lookup-limit? uint8
    +--rw rib-list* [name]
      +--rw name string
      +--rw address-family address-family-definition
      +--rw ip-rpf-check? boolean
      +--rw route-list* [route-index]
        +--rw route-index uint64
        +--rw match
          +--rw (route-type)?
            +--:(ipv4)
            |   ...
            +--:(ipv6)
            |   ...
            +--:(mpls-route)
            |   ...

```

```

| | | |--:(mac-route)
| | | | ...
| | | |--:(interface-route)
| | | | ...
|--rw nexthop
| | |--rw nexthop-id? uint32
| | |--rw sharing-flag? boolean
| | |--rw (nexthop-type)?
| | | |--:(nexthop-base)
| | | | ...
| | | |--:(nexthop-chain) {nexthop-chain}?
| | | | ...
| | | |--:(nexthop-replicates) {nexthop-replicates}?
| | | | ...
| | | |--:(nexthop-protection) {nexthop-protection}?
| | | | ...
| | | |--:(nexthop-load-balance) {nexthop-load-balance}?
| | | | ...
|--rw route-status
| | | ...
|--rw route-attributes
| | | ...
|--rw route-vendor-attributes
--rw nexthop-list* [nexthop-member-id]
    --rw nexthop-member-id uint32

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

```

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

```

```

+---:(nexthop-chain) {nexthop-chain}?
|   ...
+---:(nexthop-replicates) {nexthop-replicates}?
|   ...
+---:(nexthop-protection) {nexthop-protection}?
|   ...
+---:(nexthop-load-balance) {nexthop-load-balance}?
|   ...
+--ro output
+--ro result          uint32
+--ro reason?         string
+--ro nexthop-id?     uint32
+---x nh-delete
+---w input
|   +---w rib-name          string
|   +---w nexthop-id?      uint32
|   +---w sharing-flag?    boolean
|   +---w (nexthop-type)?
|       +---:(nexthop-base)
|           ...
|       +---:(nexthop-chain) {nexthop-chain}?
|           ...
|       +---:(nexthop-replicates) {nexthop-replicates}?
|           ...
|       +---:(nexthop-protection) {nexthop-protection}?
|           ...
|       +---:(nexthop-load-balance) {nexthop-load-balance}?
|           ...
+--ro output
+--ro result uint32
+--ro reason? string
notifications:
+---n nexthop-resolution-status-change
+--ro nexthop
+--ro nexthop-id?          uint32
+--ro sharing-flag?        boolean
+--ro (nexthop-type)?
+---:(nexthop-base)
|   ...
+---:(nexthop-chain) {nexthop-chain}?
|   ...
+---:(nexthop-replicates) {nexthop-replicates}?
|   ...
+---:(nexthop-protection) {nexthop-protection}?
|   ...
+---:(nexthop-load-balance) {nexthop-load-balance}?
|   ...
+--ro nexthop-state nexthop-state-definition

```

```

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

```

Figure 1: Overview of I2RS RIB Module Structure

2.1. RIB Capability

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

At the same time, nexthop chains can be used to specify multiple headers over a packet, before that particular packet is forwarded. Not every network device will be able to support all kinds of nexthop chains along with the arbitrary number of headers which are chained together. The RIB data model needs a way to expose the nexthop chaining capability supported by a given network device.

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

2.2. Routing Instance and Rib

A routing instance, in the context of the RIB information model, is a collection of RIBs, interfaces, and routing protocol parameters. A routing instance creates a logical slice of the router and can allow multiple different logical slices, across a set of routers, to communicate with each other. The routing protocol parameters control

the information available in the RIBs. More details about routing instance can be found in Section 2.2 of [I-D.ietf-i2rs-rib-info-model].

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

```

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

```

Figure 2: Routing Instance Structure

2.3. Route

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

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

- o ROUTE_PREFERENCE: See Section 2.3 of [I-D.ietf-i2rs-rib-info-model].
- o ACTIVE: Indicates whether a route has at least one fully resolved nexthop and is therefore eligible for installation in the FIB.
- o INSTALLED: Indicates whether the route got installed in the FIB.
- o REASON - Indicates the specific reason that caused the failure, E.g. Not authorized.

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

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


```

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

```

Figure 3: Routes Structure

2.4. Nexthop

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

- o Base: The "base" nexthop is the foundation of all other nexthop types. It includes the follow basic nexthops:
 - * nexthop-id
 - * IPv4 address

- * IPv6 address
 - * egress-interface
 - * egress-interface with IPv4 address
 - * egress-interface with IPv6 address
 - * egress-interface with MAC address
 - * logical-tunnel
 - * tunnel-encapsulation
 - * tunnel-decapsulation
 - * rib-name
- o Chain: Provide a way to perform multiple operations on a packet by logically combining them.
 - o Load-balance: Designed for load-balance case where it normally will have multiple weighted nexthops.
 - o Protection: Designed for protection scenario where it normally will have primary and standby nexthop.
 - o Replicate: Designed for multiple destinations forwarding.

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

```

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

```

Figure 4: Nexthop Structure

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

```

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

```

```

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

```

definition

```

+---:(nvgre) {nvgre-tunnel}?
+--rw nvgre-header
+--rw (nvgre-type)?
+---:(ipv4)
+--rw src-ipv4-address inet:ipv4-address
+--rw dest-ipv4-address inet:ipv4-address
+--rw protocol uint8
+--rw ttl? uint8
+--rw dscp? uint8
+---:(ipv6)
+--rw src-ipv6-address inet:ipv6-address
+--rw dest-ipv6-address inet:ipv6-address
+--rw next-header uint8
+--rw traffic-class? uint8
+--rw flow-label? inet:ipv6-flow-label
+--rw hop-limit? uint8
+--rw virtual-subnet-id uint32
+--rw flow-id? uint8
+---:(vxlan) {vxlan-tunnel}?
+--rw vxlan-header
+--rw (vxlan-type)?
+---:(ipv4)
+--rw src-ipv4-address inet:ipv4-address
+--rw dest-ipv4-address inet:ipv4-address
+--rw protocol uint8
+--rw ttl? uint8
+--rw dscp? uint8
+---:(ipv6)
+--rw src-ipv6-address inet:ipv6-address
+--rw dest-ipv6-address inet:ipv6-address
+--rw next-header uint8
+--rw traffic-class? uint8
+--rw flow-label? inet:ipv6-flow-label
+--rw hop-limit? uint8
+--rw vxlan-identifier uint32
+---:(tunnel-decapsulation-nextthop) {nextthop-tunnel}?
+--rw tunnel-decapsulation
+--rw (tunnel-type)?
+---:(ipv4) {ipv4-tunnel}?
+--rw ipv4-decapsulation
+--rw ipv4-decapsulation tunnel-decapsulation-
action-definition
+--rw ttl-action? ttl-action-definition
+---:(ipv6) {ipv6-tunnel}?
+--rw ipv6-decapsulation
+--rw ipv6-decapsulation tunnel-decapsulation-
action-definition
+--rw hop-limit-action? hop-limit-action-

```

```

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

```

Figure 5: Nexthop Base Structure

2.5. RPC Operations

This module defines the following RPC operations:

- o rib-add: Add a RIB to a routing instance. A name of the RIB, address family of the RIB and (optionally) whether the RPF check is enabled are passed as the input parameters. The output is the result of the add operation:
 - * true - success;
 - * false - failed; when failed, the i2rs agent may return the specific reason that caused the failure.
- o rib-delete: Delete a RIB from a routing instance. When a RIB is deleted, all routes installed in the RIB will be deleted. A name of the RIB is passed as the input parameter. The output is the result of the delete operation:
 - * true - success;
 - * false - failed; when failed, the i2rs agent may return the specific reason that caused the failure.
- o route-add: Add a route or a set of routes to a RIB. A RIB name, the route prefix(es), route attributes, route vendor attributes, nexthop and whether return failure details are passed as the input parameters. Before calling the route-add rpc, it is required to call the nh-add rpc to create and/or return the nexthop identifier. However, in situations when the nexthop already exists and the nexthop-id is known, this action is not expected.

The output is a combination of the route operation states while querying the appropriate node in the data tree that include:

- * success-count: the number of routes that were successfully added;
 - * failed-count: the number of the routes that failed to be added;
 - * failure-detail: shows the specific routes that failed to be added.
- o route-delete: Delete a route or a set of routes from a RIB. A name of the RIB, the route prefix(es) and whether to return failure details are passed as the input parameters. The output is a combination of route operation states that include:
- * success-count: the number of routes that were successfully deleted;
 - * failed-count: the number of the routes that failed to be deleted;
 - * failure-detail: shows the specific routes that failed to be deleted.
- o route-update: Update a route or a set of routes. A RIB name, the route prefix(es), or route attributes, or route vendor attributes, or nexthop are passed as the input parameters. The match conditions can be either route prefix(es), or route attributes, or route vendor attributes, or nexthop. The update actions include: update the nexthop, update the route attributes, update the route vendor attributes. The output is combination of the route operation states that include:
- * success-count: the number of routes that were successfully updated;
 - * failed-count: the number of the routes that failed to be updated;
 - * failure-detail: shows the specific routes that failed to be updated.
- o nh-add: Add a nexthop to a RIB. A name of the RIB and a nexthop are passed as the input parameters. The network node is required to allocate a nexthop identifier to the nexthop. The outputs include the result of the nexthop add operation.

- * true - success; when success, a nexthop identifier will be returned to the i2rs client.
 - * false - failed; when failed, the i2rs agent may return the specific reason that caused the failure.
- o nh-delete: Delete a nexthop from a RIB. A name of a RIB and a nexthop or nexthop identifier are passed as the input parameters. The output is the result of the delete operation:
- * true - success;
 - * false - failed; when failed, the i2rs agent may return the specific reason that caused the failure.

The structure tree of rpcs is shown in following figure.

```

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

```



```

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

```

```
|  +---w (nexthop-type)?  
|  ...  
+--ro output  
  +--ro result uint32  
  +--ro reason? string
```

Figure 6: RPCs Structure

2.6. Notifications

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

1. Route change notification:

- o Installed (Indicates whether the route got installed in the FIB) ;
- o Active (Indicates whether a route has at least one fully resolved nexthop and is therefore eligible for installation in the FIB) ;
- o Reason - E.g. Not authorized

2. Nexthop resolution status notification

Nexthops can be fully resolved or unresolved.

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

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

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

- o from the active state to the inactive state
- o from the inactive state to the active state

- o from the installed state to the uninstalled state
- o from the uninstalled state to the installed state

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

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

notifications:

```

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

```

Figure 7: Notifications Structure

3. YANG Modules

```
<CODE BEGINS> file "ietf-i2rs-rib@2018-04-23.yang"

module ietf-i2rs-rib {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-i2rs-rib";
  prefix "iir";

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

  import ietf-interfaces {
    prefix if;
    reference "RFC 8344";
  }

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

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

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

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

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

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

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

    Editor:     Nitin Bahadur
                <mailto:nitin_bahadur@yahoo.com>";
  description
```

```
"This module defines a YANG data model for
Routing Information Base (RIB) that aligns
with the I2RS RIB information model.
Copyright (c) <2018> IETF Trust and the persons
identified as authors of the code. All rights reserved.";
revision "2018-04-23" {
  description "initial revision";
  reference "RFC XXXX: draft-ietf-i2rs-data-model-10";
  // RFC Ed.: replace XXXX with actual RFC number and remove
  // this note
}

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

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

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

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

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

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

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

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

feature vxlan-tunnel {
  description
    "This feature means that a node supports
    VXLAN tunnel encapsulation capability.";
  reference "RFC7348";
}

feature gre-tunnel {
  description
    "This feature means that a node supports
    GRE tunnel encapsulation capability.";
  reference "RFC2784";
}

feature nvgre-tunnel {
  description
    "This feature means that a node supports
    NvGRE tunnel encapsulation capability.";
  reference "RFC7637";
}

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

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

```
        stack with new label.";
    }

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

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

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

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

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

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

    identity ipv6-decapsulation {
        base "tunnel-decapsulation-action";
        description
            "IPv6 tunnel decapsulation.";
```

```
}

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

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

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

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

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

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

typedef ttl-action-definition {
    type identityref {
```



```
    base "ttl-action";
  }
  description
    "TTL action definition.";
}

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

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

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

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

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

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

identity discard-with-error {
```

```
    base "special-nexthop";
    description
        "This indicates that the network
        device should drop the packet,
        increment a drop counter and send
        back an appropriate error message
        (like ICMP error).";
}

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

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

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

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

identity match-ip-src {
```

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

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

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

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

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

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

identity ieee-mac-address-family {
```

```
    base "address-family";
    description
        "MAC RIB address family.";
}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
}

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

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

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

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

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

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

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

```
identity unresolved {
  base "nexthop-state";
  description
    "Unresolved nexthop state.";
}

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

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

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

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

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

//Route change reason identities

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

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

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

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

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

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

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



```
        nexthops that have the same preference, an
        implementation of i2rs client should select N
        nexthops and download them, as for how to select
        the nexthops is left to the implementations.";
    }

typedef nexthop-lb-weight-definition {
    type uint8 {
        range "1..99";
    }
    description
        "Nexthop-lb-weight is used for load-balancing.
        Each list member SHOULD 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 sum of 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: The weight of 0 is specially because of
        historical reasons. It's typically used in
        hardware devices to signify ECMP";
}

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

//Groupings
grouping route-prefix {
    description
        "The common attributes used for all types of route prefix.";
    leaf route-index {
        type uint64 ;
    }
}
```

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

```

        description
            "The IPv4 destination address of the match.";
    }
    leaf src-ipv4-prefix {
        type inet:ipv4-prefix;
        mandatory true;
        description
            "The IPv4 source address of the match";
    }
    }
    }
    }
}
case ipv6 {
    description
        "IPv6 route case.";
    container ipv6 {
        description
            "IPv6 route match.";
        choice ip-route-match-type {
            description
                "IP route match type options:
                match source, or
                match destination, or
                match source and destination.";
            case dest-ipv6-address {
                leaf dest-ipv6-prefix {
                    type inet:ipv6-prefix;
                    mandatory true;
                    description
                        "An IPv6 destination address as the match.";
                }
            }
            case src-ipv6-address {
                leaf src-ipv6-prefix {
                    type inet:ipv6-prefix;
                    mandatory true;
                    description
                        "An IPv6 source address as the match.";
                }
            }
            case dest-src-ipv6-address {
                container dest-src-ipv6-address {
                    description
                        "A combination of an IPv6 source and
                        an IPv6 destination address as the match.";
                    leaf dest-ipv6-prefix {

```

```

        type inet:ipv6-prefix;
        mandatory true;
        description
            "The IPv6 destination address of the match";
    }
    leaf src-ipv6-prefix {
        type inet:ipv6-prefix;
        mandatory true;
        description
            "The IPv6 source address of the match.";
    }
}
}
}
}
}
}
case mpls-route {
    description
        "MPLS route case.";
    leaf mpls-label {
        type uint32 ;
        mandatory true;
        description
            "The label used for matching.";
    }
}
case mac-route {
    description
        "MAC route case.";
    leaf mac-address {
        type yang:mac-address;
        mandatory true;
        description
            "The MAC address used for matching.";
    }
}
case interface-route {
    description
        "Interface route case.";
    leaf interface-identifier {
        type if:interface-ref;
        mandatory true;
        description
            "The interface used for matching.";
    }
}
}
}
}
}

```

```
}

grouping route {
  description
    "The common attributes used for all types of routes.";
  uses route-prefix;
  container nexthop {
    description
      "The nexthop of the route.";
    uses nexthop;
  }
  //In the information model, it is called route-statistic
  container route-status {
    description
      "The status information of the route.";
    leaf route-state {
      type route-state-definition;
      config false;
      description
        "Indicate a route's state: Active or Inactive.";
    }
    leaf route-installed-state {
      type route-installed-state-definition;
      config false;
      description
        "Indicate that a route's installed states:
        Installed or uninstalled.";
    }
    leaf route-reason {
      type route-change-reason-definition;
      config false;
      description
        "Indicate the reason that caused the route change.";
    }
  }
  container route-attributes {
    description
      "Route attributes.";
    uses route-attributes;
  }
  container route-vendor-attributes {
    description
      "Route vendor attributes.";
    uses route-vendor-attributes;
  }
}

grouping nexthop-list {
```

```
description
  "A generic nexthop list.";
list nexthop-list {
  key "nexthop-member-id";
  description
    "A list of nexthops.";
  leaf nexthop-member-id {
    type uint32;
    mandatory true;
    description
      "A nexthop identifier that points
      to a nexthop list member.
      A nexthop list member is a nexthop.";
  }
}

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

grouping nexthop-list-w {
  description
```

```
    "A nexthop list with weight parameter.";
list nexthop-list {
  key "nexthop-member-id";
  description
    "A list of nexthop.";
  leaf nexthop-member-id {
    type uint32;
    mandatory true;
    description
      "A nexthop identifier that points
       to a nexthop list member.
       A nexthop list member is a nexthop.";
  }
  leaf nexthop-lb-weight {
    type nexthop-lb-weight-definition;
    mandatory true;
    description
      "The weight of a nexthop of
       the load balance nexthops.";
  }
}

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

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

grouping nexthop-base {
    description
        "The base nexthop.";
    choice nexthop-base-type {
        description
            "Nexthop base type options.";
        case special-nexthop {
            leaf special {
                type special-nexthop-definition;
                description

```



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

```
case egress-interface-ipv6-nexthop {
  container egress-interface-ipv6-address {
    leaf outgoing-interface {
      type if:interface-ref;
      mandatory true;
      description
        "Name of the outgoing interface.";
    }
    leaf ipv6-address {
      type inet:ipv6-address;
      mandatory true;
      description
        "The nexthop points to an interface with
         an IPv6 address.";
    }
    description
      "The nexthop is an egress-interface and an IP
       address. This can be used in cases e.g. where
       the IP address is a link-local address.";
  }
}
case egress-interface-mac-nexthop {
  container egress-interface-mac-address {
    leaf outgoing-interface {
      type if:interface-ref;
      mandatory true;
      description
        "Name of the outgoing interface.";
    }
    leaf ieee-mac-address {
      type yang:mac-address;
      mandatory true;
      description
        "The nexthop points to an interface with
         a specific mac-address.";
    }
    description
      "The egress interface must be an Ethernet
       interface. Address resolution is not required
       for this nexthop.";
  }
}
case tunnel-encap-nexthop {
  if-feature nexthop-tunnel;
  container tunnel-encap {
    uses tunnel-encap;
    description
      "This can be an encapsulation representing an IP
```

```

        tunnel or MPLS tunnel or others as defined in info
        model. An optional egress interface can be chained
        to the tunnel encapsulation to indicate which
        interface to send the packet out on. The egress
        interface is useful when the network device
        contains Ethernet interfaces and one needs to
        perform address resolution for the IP packet.";
    }
}
case tunnel-decapsulation-nexthop {
    if-feature nexthop-tunnel;
    container tunnel-decapsulation {
        uses tunnel-decapsulation;
        description
            "This is to specify the decapsulation of a tunnel header.";
    }
}
case logical-tunnel-nexthop {
    if-feature nexthop-tunnel;
    container logical-tunnel {
        uses logical-tunnel;
        description
            "This can be a MPLS LSP or a GRE tunnel (or others
            as defined in this document), that is represented
            by a unique identifier (e.g. name).";
    }
}
case rib-name-nexthop {
    leaf rib-name {
        type string;
        description
            "A nexthop pointing to a RIB indicates that the
            route lookup needs to continue in the specified
            RIB. This is a way to perform chained lookups.";
    }
}
case nexthop-identifier {
    leaf nexthop-ref {
        type nexthop-ref;
        mandatory true;
        description
            "A nexthop reference that points to a nexthop.";
    }
}
}
}
grouping route-vendor-attributes {

```

```
    description
      "Route vendor attributes.";
  }

  grouping logical-tunnel {
    description
      "A logical tunnel that is identified
       by a type and a tunnel name.";
    leaf tunnel-type {
      type tunnel-type-definition;
      mandatory true;
      description
        "A tunnel type.";
    }
    leaf tunnel-name {
      type string;
      mandatory true;
      description
        "A tunnel name that points to a logical tunnel.";
    }
  }

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

```
    leaf dscp {
      type uint8;
      description
        "The DSCP field of the header.";
    }
  }

  grouping ipv6-header {
    description
      "The IPv6 header encapsulation information.";
    leaf src-ipv6-address {
      type inet:ipv6-address;
      mandatory true;
      description
        "The source IP address of the header.";
    }
    leaf dest-ipv6-address {
      type inet:ipv6-address;
      mandatory true;
      description
        "The destination IP address of the header.";
    }
    leaf next-header {
      type uint8;
      mandatory true;
      description
        "The next header of the IPv6 header.";
    }
    leaf traffic-class {
      type uint8;
      description
        "The traffic class value of the header.";
    }
    leaf flow-label {
      type inet:ipv6-flow-label;
      description
        "The flow label of the header.";
    }
    leaf hop-limit {
      type uint8 {
        range "1..255";
      }
      description
        "The hop limit of the header.";
    }
  }
}

grouping nvgre-header {
```

```
description
  "The NvGRE header encapsulation information.";
choice nvgre-type {
  description
    "NvGRE can use either IPv4
    or IPv6 header for encapsulation.";
  case ipv4 {
    uses ipv4-header;
  }
  case ipv6 {
    uses ipv6-header;
  }
}
leaf virtual-subnet-id {
  type uint32;
  mandatory true;
  description
    "The subnet identifier of the NvGRE header.";
}
leaf flow-id {
  type uint8;
  description
    "The flow identifier of the NvGRE header.";
}
}

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

```
grouping gre-header {
  description
    "The GRE encapsulation header information.";
  choice dest-address-type {
    description
      "GRE options: IPv4 and IPv6";
    case ipv4 {
      leaf ipv4-dest {
        type inet:ipv4-address;
        mandatory true;
        description
          "The destination IP address of the GRE header.";
      }
    }
    case ipv6 {
      leaf ipv6-dest {
        type inet:ipv6-address;
        mandatory true;
        description
          "The destination IP address of the GRE header.";
      }
    }
  }
  leaf protocol-type {
    type uint16;
    mandatory true;
    description
      "The protocol type of the GRE header.";
  }
  leaf key {
    type uint64;
    description
      "The GRE key of the GRE header.";
  }
}

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

```
}
choice label-actions {
  description
    "Label action options.";
  case label-push {
    container label-push {
      description
        "Label push operation.";
      leaf label {
        type uint32;
        mandatory true;
        description
          "The label to be pushed.";
      }
      leaf s-bit {
        type boolean;
        description
          "The s-bit of the label to be pushed. ";
      }
      leaf tc-value {
        type uint8;
        description
          "The traffic class value of the label to be pushed.";
      }
      leaf ttl-value {
        type uint8;
        description
          "The TTL value of the label to be pushed.";
      }
    }
  }
}
case label-swap {
  container label-swap {
    description
      "Label swap operation.";
    leaf in-label {
      type uint32;
      mandatory true;
      description
        "The label to be swapped.";
    }
    leaf out-label {
      type uint32;
      mandatory true;
      description
        "The out MPLS label.";
    }
    leaf ttl-action {
```



```

        type ttl-action-definition;
        description
            "The label ttl actions:
            - No-action, or
            - Copy to inner label, or
            - Decrease (the in-label) by 1 and
              copy to the out-label.";
    }
}
}
}
}
}
}

```

```

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

```

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

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

```
case ipv6 {
  if-feature ipv6-tunnel;
  container ipv6-decapsulation {
    description
      "IPv6 decapsulation.";
    leaf ipv6-decapsulation {
      type tunnel-decapsulation-action-definition;
      mandatory true;
      description
        "IPv6 decapsulation operations.";
    }
    leaf hop-limit-action {
      type hop-limit-action-definition;
      description
        "The hop limit actions:
         no-action or copy to inner header.";
    }
  }
}
case mpls {
  if-feature mpls-tunnel;
  container label-pop {
    description
      "MPLS decapsulation.";
    leaf label-pop {
      type mpls-label-action-definition;
      mandatory true;
      description
        "Pop a label from the label stack.";
    }
    leaf ttl-action {
      type ttl-action-definition;
      description
        "The label ttl action.";
    }
  }
}
}

grouping route-attributes {
  description
    "Route attributes.";
  leaf route-preference {
    type uint32;
    mandatory true;
    description
      "ROUTE_PREFERENCE: This is a numerical value that
```

```
        allows for comparing routes from different
        protocols.  Static configuration is also
        considered a protocol for the purpose of this
        field.  It is also known as administrative-distance.
        The lower the value, the higher the preference.";
    }
    leaf local-only {
        type boolean ;
        mandatory true;
        description
            "Indicate whether the attributes is local only.";
    }
    container address-family-route-attributes{
        description
            "Address family related route attributes.";
        choice route-type {
            description
                "Address family related route attributes.";
            case ip-route-attributes {
            }
            case mpls-route-attributes {
            }
            case ethernet-route-attributes {
            }
        }
    }
}

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

```
        interface list contains a list of identifiers, with
        each identifier uniquely identifying an interface.";
    leaf name {
        type if:interface-ref;
        description
            "A reference to the name of a network layer interface.";
    }
}
leaf router-id {
    type yang:dotted-quad;
    description
        "Router ID - 32-bit number in the form of a dotted quad.";
}
leaf lookup-limit {
    type uint8;
    description
        "A limit on how many levels of a lookup can be performed.";
}
list rib-list {
    key "name";
    description
        "A list of RIBs that are associated with the routing
        instance.";
    leaf name {
        type string;
        mandatory true;
        description
            "A reference to the name of each RIB.";
    }
    leaf address-family {
        type address-family-definition;
        mandatory true;
        description
            "The address family of a RIB.";
    }
    leaf ip-rpf-check {
        type boolean;
        description
            "Each RIB can be optionally associated with a
            ENABLE_IP_RPF_CHECK attribute that enables Reverse
            path forwarding (RPF) checks on all IP routes in that
            RIB. Reverse path forwarding (RPF) check is used to
            prevent spoofing and limit malicious traffic.";
    }
}
list route-list {
    key "route-index";
    description
        "A list of routes of a RIB.";
```

```
        uses route;
    }
    // This is a list that maintains the nexthops added to the RIB.
    uses nexthop-list;
}

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

```
    }  
  }  
}  
  
rpc rib-delete {  
  description  
    "To delete a RIB from a routing instance.  
    After deleting the RIB, all routes installed  
    in the RIB will be deleted as well.";  
  input {  
    leaf name {  
      type string;  
      mandatory true;  
      description  
        "A reference to the name of the RIB  
        that is to be deleted.";  
    }  
  }  
  output {  
    leaf result {  
      type boolean;  
      mandatory true;  
      description  
        "Return the result of the rib-delete operation.  
        true - success;  
        false - failed";  
    }  
    leaf reason {  
      type string;  
      description  
        "The specific reason that caused failure.";  
    }  
  }  
}  
  
grouping route-operation-state {  
  description  
    "Route operation state.";  
  leaf success-count {  
    type uint32;  
    mandatory true;  
    description  
      "The numbers of routes that are successfully  
      added/deleted/updated.";  
  }  
  leaf failed-count {  
    type uint32;  
    mandatory true;  
  }  
}
```

```

    description
        "The numbers of the routes that are failed
        to be added/deleted/updated.";
    }
    container failure-detail {
        description
            "The failure detail reflects the reason why a route
            operation fails. It is a array that includes the route
            index and error code of the failed route.";
        list failed-routes {
            key "route-index";
            description
                "The list of failed routes.";
            leaf route-index {
                type uint32;
                description
                    "The route index of the failed route.";
            }
            leaf error-code {
                type uint32;
                description
                    "The error code that reflects the failure reason.
                    0 - Reserved.
                    1 - Trying to add a repeat route;
                    2 - Trying to delete or update a route that is not exist;
                    3 - Malformed route attribute;
                    ";
            }
        }
    }
}

rpc route-add {
    description
        "To add a route or a list of route to a RIB";
    input {
        leaf return-failure-detail {
            type boolean;
            default false;
            description
                "Whether return the failure detail.
                true - return the failure detail;
                false - do not return the failure detail;
                the default is false.";
        }
        leaf rib-name {
            type string;
            mandatory true;
        }
    }
}

```



```
    description
      "A reference to the name of a RIB.";
  }
  container routes {
    description
      "The routes to be added to the RIB.";
    list route-list {
      key "route-index";
      description
        "The list of routes to be added.";
      uses route-prefix;
      container route-attributes {
        uses route-attributes;
        description
          "The route attributes.";
      }
      container route-vendor-attributes {
        if-feature route-vendor-attributes;
        uses route-vendor-attributes;
        description
          "The route vendor attributes.";
      }
      container nexthop {
        uses nexthop;
        description
          "The nexthop of the added route.";
      }
    }
  }
}
output {
  uses route-operation-state;
}
}

rpc route-delete {
  description
    "To delete a route or a list of route from a RIB";
  input {
    leaf return-failure-detail {
      type boolean;
      default false;
      description
        "Whether return the failure detail.
         true  - return the failure detail;
         false - do not return the failure detail;
         the default is false.";
    }
  }
}
```

```
    leaf rib-name {
      type string;
      mandatory true;
      description
        "A reference to the name of a RIB.";
    }
    container routes {
      description
        "The routes to be added to the RIB.";
      list route-list {
        key "route-index";
        description
          "The list of routes to be deleted.";
        uses route-prefix;
      }
    }
  }
  output {
    uses route-operation-state;
  }
}

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

```
    case update-route-vendor-attributes {
      container updated-route-vendor-attr {
        uses route-vendor-attributes;
        description
          "The vendor route attributes used for updating.";
      }
    }
  }
}

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

```
case match-route-prefix {
  description
    "Update the routes that match route
    prefix(es) condition.";
  container input-routes {
    description
      "The matched routes to be updated.";
    list route-list {
      key "route-index";
      description
        "The list of routes to be updated.";
      uses route-prefix;
      uses route-update-options;
    }
  }
}
case match-route-attributes {
  description
    "Update the routes that match the
    route attributes condition.";
  container input-route-attributes {
    description
      "The route attributes are used for matching.";
    uses route-attributes;
  }
  container update-parameters {
    description
      "Update options:
      1. update the nexthop
      2. update the route attributes
      3. update the route-vendor-attributes.";
    uses route-update-options;
  }
}
case match-route-vendor-attributes {
  if-feature route-vendor-attributes;
  description
    "Update the routes that match the
    vendor attributes condition";
  container input-route-vendor-attributes {
    description
      "The vendor route attributes are used for matching.";
    uses route-vendor-attributes;
  }
  container update-parameters-vendor {
    description
      "Update options:
      1. update the nexthop
```

```

        2. update the route attributes
        3. update the route-vendor-attributes.";
    uses route-update-options;
}
}
case match-nexthop {
    description
        "Update the routes that match the nexthop.";
    container input-nexthop {
        description
            "The nexthop used for matching.";
        uses nexthop;
    }
    container update-parameters-nexthop {
        description
            "Update options:
            1. update the nexthop
            2. update the route attributes
            3. update the route-vendor-attributes.";
        uses route-update-options;
    }
}
}
}
output {
    uses route-operation-state;
}
}

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

```

```
    }
    uses nexthop;
  }
  output {
    leaf result {
      type boolean;
      mandatory true;
      description
        "Return the result of the rib-add operation.
         true  - success;
         false - failed.";
    }
    leaf reason {
      type string;
      description
        "The specific reason that caused the failure.";
    }
    leaf nexthop-id {
      type uint32;
      description
        "A nexthop identifier that is allocated to the nexthop.";
    }
  }
}

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

```
        description
            "The specific reason that caused the failure.";
    }
}

/*Notifications*/
notification nexthop-resolution-status-change {
    description
        "Nexthop resolution status (resolved/unresolved)
        notification.";
    container nexthop{
        description
            "The nexthop.";
        uses nexthop;
    }
    leaf nexthop-state {
        type nexthop-state-definition;
        mandatory true;
        description
            "Nexthop resolution status (resolved/unresolved)
            notification.";
    }
}

notification route-change {
    description
        "Route change notification.";
    leaf rib-name {
        type string;
        mandatory true;
        description
            "A reference to the name of a RIB.";
    }
    leaf address-family {
        type address-family-definition;
        mandatory true;
        description
            "The address family of a RIB.";
    }
    uses route-prefix;
    leaf route-installed-state {
        type route-installed-state-definition;
        mandatory true;
        description
            "Indicates whether the route got installed in the FIB.";
    }
    leaf route-state {
```

```

    type route-state-definition;
    mandatory true;
    description
        "Indicates whether a route is active or inactive.";
}
list route-change-reasons {
    key "route-change-reason";
    description
        "The reasons that cause the route change. A route
        change that may result from several reasons. For
        example, a nexthop becoming resolved will make a
        route A active which is of better preference than
        a currently active route B, which results in the
        route A being installed";
    leaf route-change-reason {
        type route-change-reason-definition;
        mandatory true;
        description
            "The reason that caused the route change.";
    }
}
}
}

```

<CODE ENDS>

4. IANA Considerations

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

```

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

```

This document requests to register a YANG module in the "YANG Module Names registry" [RFC7950]:

```

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

```


5. Security Considerations

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

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

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

- o RIB: A malicious client could attempt to remove a RIB from a routing instance, for example in order to sabotage the services provided by the RIB, or to add a RIB to a routing instance, hence to inject unauthorized traffic into the nexthop.
- o route: A malicious client could attempt to remove or add a route from/to a RIB, for example in order to sabotage the services provided by the RIB.
- o nexthop: A malicious client could attempt to remove or add a nexthop from/to RIB, which may lead to suboptimal path and

degradation of service levels as well as possibly disruption of service.

6. Contributors

The following individuals also contribute to this document.

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

7. Acknowledgements

The authors would like to thank Chris Bowers, John Scudder, Tom Petch, Mike McBride and Ebben Aries for his review, suggestion and comments to this document.

8. References

8.1. Normative References

- [I-D.ietf-i2rs-rib-info-model] Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-17 (work in progress), May 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.

- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <<https://www.rfc-editor.org/info/rfc8344>>.

8.2. Informative References

- [I-D.ietf-i2rs-usecase-reqs-summary] Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", draft-ietf-i2rs-usecase-reqs-summary-03 (work in progress), November 2016.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<https://www.rfc-editor.org/info/rfc2784>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014, <<https://www.rfc-editor.org/info/rfc7348>>.

- [RFC7637] Garg, P., Ed. and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <<https://www.rfc-editor.org/info/rfc7637>>.
- [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>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.

Authors' Addresses

Lixing Wang
Individual

Email: wang_little_star@sina.com

Mach(Guoyi) Chen
Huawei

Email: mach.chen@huawei.com

Amit Dass
Ericsson

Email: amit.dass@ericsson.com

Hariharan Ananthakrishnan
Packet Design

Email: hari@packetdesign.com

Sriganesh Kini
Individual

Email: sriganeshkini@gmail.com

Nitin Bahadur
Bracket Computing

Email: nitin_bahadur@yahoo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 8, 2018

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

J. Medved
Cisco
May 7, 2018

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.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

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

Table of Contents

1. Introduction	3
1.1. Conventions used in this document	5
2. RIB data	5
2.1. RIB definition	6
2.2. Routing instance	6
2.3. Route	7
2.4. Nexthop	9
2.4.1. Base nexthop	12
2.4.2. Derived nexthops	13
2.4.3. Nexthop indirection	15
3. Reading from the RIB	15
4. Writing to the RIB	15
5. Notifications	16
6. RIB grammar	16
6.1. Nexthop grammar explained	19
7. Using the RIB grammar	19
7.1. Using route preference	19
7.2. Using different nexthops types	20
7.2.1. Tunnel nexthops	20
7.2.2. Replication lists	20
7.2.3. Weighted lists	21
7.2.4. Protection	21
7.2.5. Nexthop chains	22
7.2.6. Lists of lists	23
7.3. Performing multicast	24
8. RIB operations at scale	25
8.1. RIB reads	25
8.2. RIB writes	25
8.3. RIB events and notifications	25
9. Security Considerations	25
10. IANA Considerations	26
11. Acknowledgements	26
12. References	26
12.1. Normative References	26
12.2. Informative References	27
Authors' Addresses	28

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.

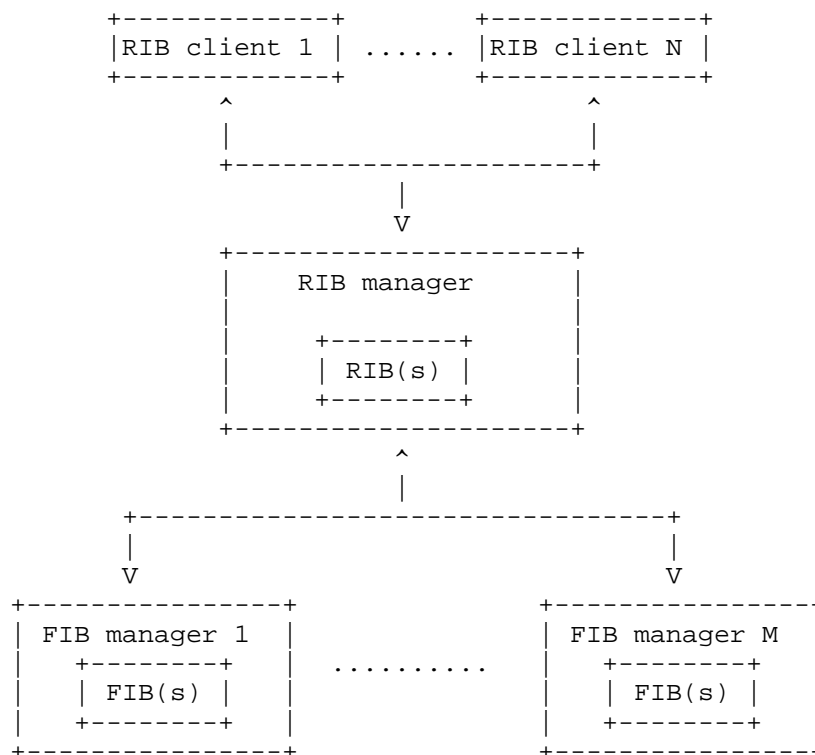


Figure 1: RIB manager, RIB clients, and FIB managers

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.

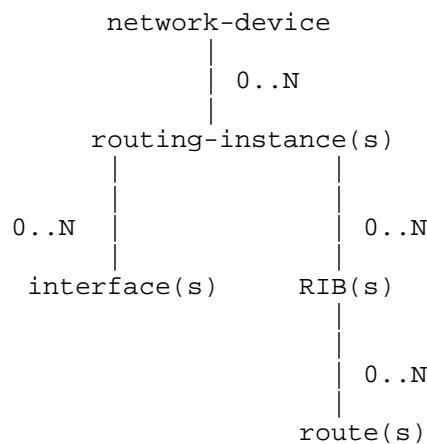


Figure 2: RIB model

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.

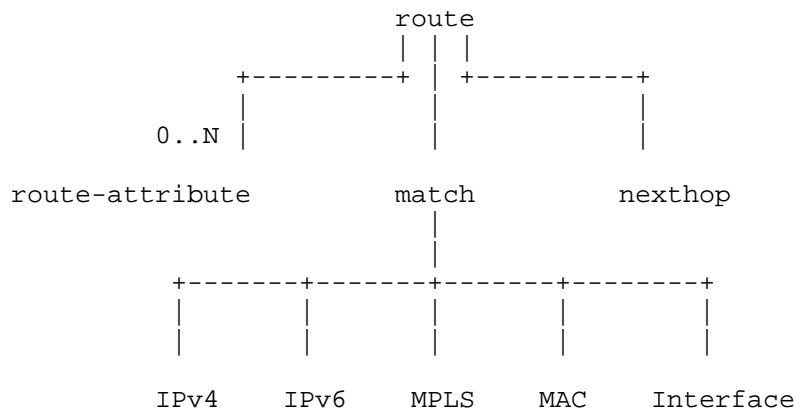


Figure 3: Route model

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.

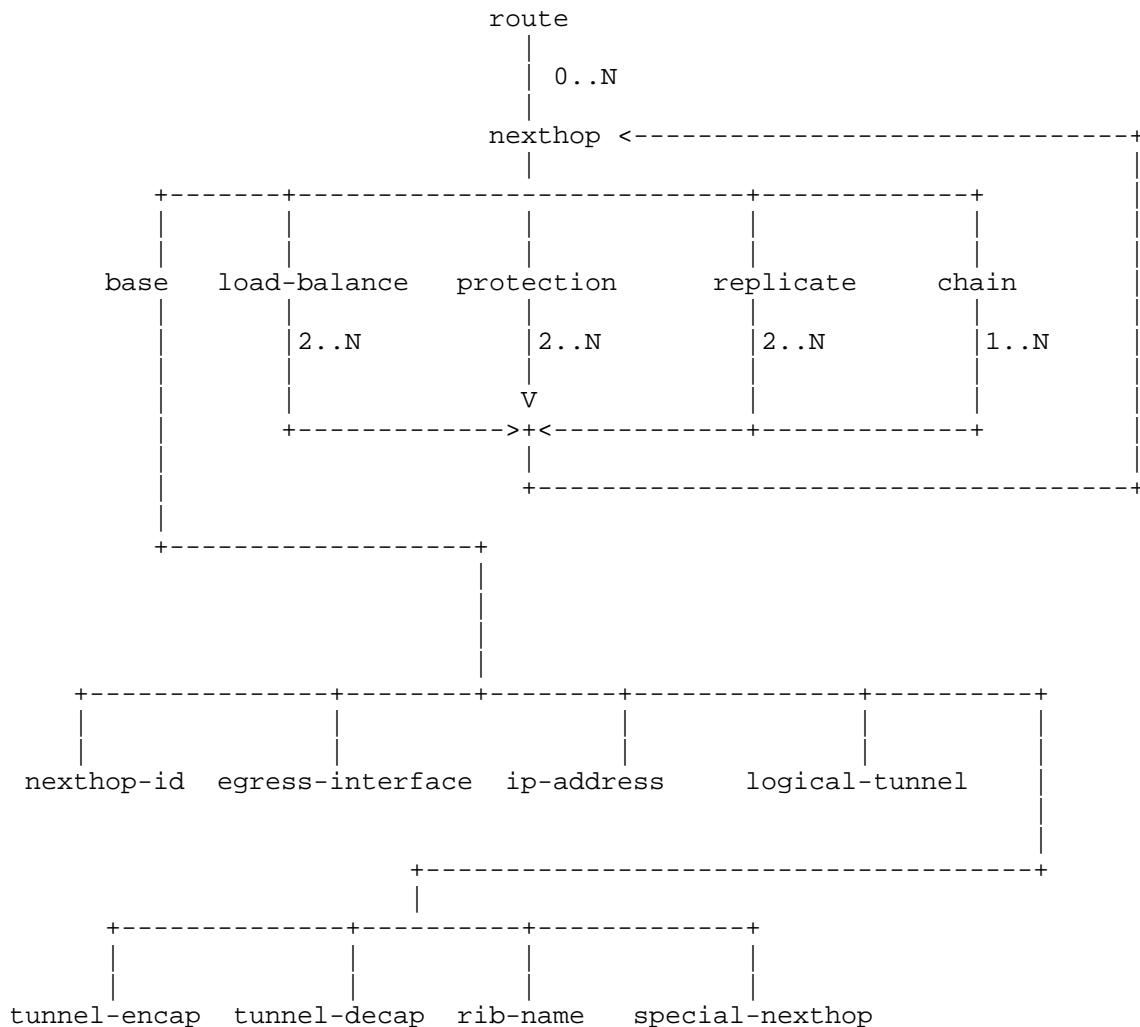


Figure 4: Nexthop model

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 its 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_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_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_LOAD_BALANCE>
    (<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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 13, 2016

A. Clemm
J. Medved
Cisco
R. Varga
T. Tkacik
Pantheon Technologies SRO
X. Liu
Ericsson
I. Bryskin
Huawei
A. Guo
Adva Optical
H. Ananthakrishnan
Packet Design
N. Bahadur
Bracket Computing
V. Beeram
Juniper Networks
December 11, 2015

A YANG Data Model for Layer 3 Topologies
draft-ietf-i2rs-yang-l3-topology-01.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 <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 June 13, 2016.

Copyright Notice

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

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Definitions and Acronyms	4
3. Model overview	5
3.1. Model structure	5
3.2. Layer 3 Unicast - IGP	6
3.3. OSPF Topology	7
3.4. IS-IS Topology	9
4. Layer 3 Unicast IGP Topology YANG Module	10
5. OSPF Topology YANG Module	19
6. ISIS Topology YANG Module	26
7. Security Considerations	31
8. Contributors	31
9. Acknowledgements	32
10. References	32
10.1. Normative References	32
10.2. Informative References	33
Authors' Addresses	33

1. Introduction

This document introduces a YANG [RFC6020] [RFC6991] [I-D.draft-ietf-netmod-rfc6020bis-09] data model for Layer 3 network topologies. 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]. An earlier revision of that Internet Draft contained not just the general model for network topologies, but also the model for layer 3 network topologies that is being specified here. However, we decided to "split" the earlier draft to separate the truly general aspects of a topology data model, which apply to any type of topology, from the application of this model to a particular domain, here: a Layer 3 network.

Specific topology types that are covered in this document include Layer 3 Unicast IGP, IS-IS [RFC1195], and OSPF [RFC2178]. In addition, this documents defines a set of traffic engineering extensions.

There are multiple applications for such a data model and 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.

There are several reasons to choose YANG to define the data model. Data defined using YANG can be exposed by a server to client applications and controllers via Netconf [RFC6241] or via a ReST-like Interface [I-D.draft-ietf-netconf-restconf] [I-D.draft-ietf-netmod-yang-json]. The fact that it can be used with different protocols and interfaces provides for a degree of "future-proofing" of model implementations. Also, YANG can serve as the basis for model-driven toolchains, such as used in the Open Daylight project.

The data model is defined in several YANG modules:

- o Module "ietf-l3-unicast-igp-topology" defines a model for Layer 3 Unicast IGP topologies. To do so, it augments general network topology model defined in [I-D.draft-ietf-i2rs-yang-network-topol] with information specific to Layer 3 Unicast IGP. In doing so, it also illustrates the extension patterns associated with extending respectively augmenting the general topology model to meet the needs of a specific topology.
- o Module "ietf-ospf-topology" defines a topology model for OSPF, building on and extending the Layer 3 Unicast IGP topology model. It serves as an example of how the general topology model can be refined across multiple levels.
- o Module "ietf-isis-topology" defines a topology model for IS-IS, again building on and extending the Layer 3 Unicast IGP topology model.

Information that is kept in the Traffic Engineering Database (TED) is specified in a separate model and outside the scope of this specification.

2. Definitions and Acronyms

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

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

HTTP: Hyper-Text Transfer Protocol

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

LSP: Label Switched Path

NETCONF: Network Configuration Protocol

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

URI: Uniform Resource Identifier

ReST: Representational State Transfer, a style of stateless interface and protocol that is generally carried over HTTP

SRLG: Shared Risk Link Group

TED: Traffic Engineering Database

YANG: A data definition language for NETCONF

3. Model overview

This section provides an overview of the Layer 3 network topology model.

3.1. Model structure

The network topology model is defined by the following YANG modules, whose relationship is roughly depicted in the figure below. The base network topology is included in the diagram for completeness.

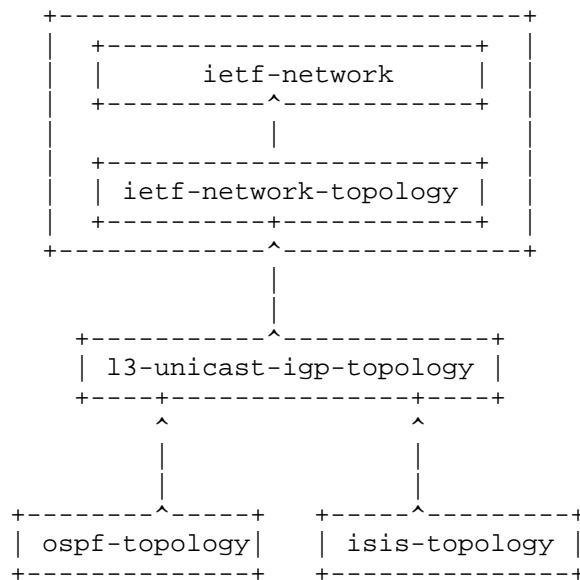


Figure 1: Overall model structure

YANG modules `ietf-network` and `ietf-network-topology` collectively define the basic network topology model. YANG module `ietf-l3-unicast-igp-topology` augments those models with additional definitions needed to represent Layer 3 Unicast IGP topologies. This module in turn is augmented by YANG modules with additional definitions for OSPF and for IS-IS topologies, `ietf-ospf-topology` and `ietf-isis-topology`, respectively.

3.2. Layer 3 Unicast - IGP

The Layer 3 Unicast IGP topology model is defined by YANG module "ietf-l3-unicast-igp-topology". The model is depicted in the following diagram. Brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parantheses enclose choice and case nodes. Notifications are not depicted. The prefix "nt:" refers to the YANG module for network topology.

```

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

```

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

- o A new network topology type is introduced, l3-unicast-igp-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 an IGP name, 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 that are associated with specific IGP topologies, 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 an IGP 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 or identifier.

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.

3.3. OSPF Topology

OSPF is the next type of topology represented in the model. OSPF represents a particular type of Layer 3 Unicast IGP. Accordingly, this time the Layer 3 Unicast IGP topology model needs to be extended. The corresponding extensions are introduced in a separate YANG module "ietf-ospf-topology", whose structure is depicted in the following diagram. For the most part, this module augments "ietf-l3-unicast-igp-topology". Like before, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses enclose choice and case nodes. A "+" at the end of a line indicates a line break. Notifications respectively augmentations of notifications are not depicted.

```

module: ietf-ospf-topology
augment /nw:networks/nw:network/nw:network-types/+
|   l3t:l3-unicast-igp-topology:
  +--rw ospf!
augment /nw:networks/nw:network/l3t:igp-topology-attributes:
  +--rw ospf-topology-attributes
    +--rw area-id?   area-id
augment /nw:networks/nw:network/nw:node/l3t:igp-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
    +--rw multi-topology-id*  uint8
    +--rw capabilities?      bits
augment /nw:networks/nw:network/nt:link/l3t:igp-link-attributes:
  +--rw ospf-link-attributes
    +--rw multi-topology-id?  uint8
augment /nw:networks/nw:network/nw:node/l3t:igp-node-attributes/+
|   l3t:prefix:
  +--rw ospf-prefix-attributes
    +--rw forwarding-address?  inet:ipv4-address

```

The module augments "ietf-l3-unicast-igp-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 igp-topology-attributes of the ietf-l3-unicast-igp-topology module. The attributes include an OSPF area-id identifying the OSPF area.
- o Additional data objects for nodes are introduced by augmenting the igp-node-attributes of the l3-unicast-igp-topology module. New objects include router-type, dr-interface-id for pseudonodes, list of multi-topology-ids, ospf node capabilities, and traffic engineering attributes.
- o Links are augmented with a multi-topology-id and traffic engineering link attributes.

- o Prefixes are augmented with OSPF specific forwarding address.

In addition, the module extends IGP node, link and prefix notifications with OSPF attributes.

3.4. IS-IS Topology

IS-IS is another type of Layer 3 Unicast IGP. Like OSPF topology, IS-IS topology is defined in a separate module, "ietf-isis-topology", which augments "ietf-l3-unicast-igp-topology". The structure is depicted in the following diagram. Like before, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses enclose choice and case nodes. A "+" at the end of a line indicates a line break. Notifications are not depicted.

```

module: ietf-isis-topology
augment /nw:networks/nw:network/nw:network-types/+
  | l3t:l3-unicast-igp-topology:
  +--rw isis!
augment /nw:networks/nw:network/l3t:igp-topology-attributes:
  +--rw isis-topology-attributes
    +--rw net? iso-net-id
augment /nw:networks/nw:network/nw:node/l3t:igp-node-attributes:
  +--rw isis-node-attributes
    +--rw iso
      | +--rw iso-system-id? iso-system-id
      | +--rw iso-pseudonode-id? iso-pseudonode-id
    +--rw net* iso-net-id
    +--rw multi-topology-id* uint8
    +--rw (router-type)?
      +--:(level-2)
      | +--rw level-2? empty
      +--:(level-1)
      | +--rw level-1? empty
      +--:(level-1-2)
      | +--rw level-1-2? empty
augment /nw:networks/nw:network/nt:link/l3t:igp-link-attributes:
  +--rw isis-link-attributes
    +--rw multi-topology-id? uint8

```

The module augments the ietf-l3-unicast-igp-topology as follows:

- o A new topology type is introduced for isis.
- o Additional topology attributes are introduced in a new grouping which augments "igp-topology-attributes" of the ietf-l3-unicast-

igp-topology module. The attributes include an ISIS NET-id identifying the area.

- o Additional data objects for nodes are introduced by augmenting "igp-node-attributes" of the ietf-l3-unicast-igp-topology module. New objects include router-type, iso-system-id to identify the router, a list of multi-topology-id, a list of NET ids, and traffic engineering attributes.
- o Links are augmented with multi-topology-id and traffic engineering link attributes.

In addition, the module augments IGP nodes and links with ISIS attributes.

4. Layer 3 Unicast IGP Topology YANG Module

```
<CODE BEGINS> file "ietf-l3-unicast-igp-topology@2015-12-11.yang"
module ietf-l3-unicast-igp-topology {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-l3-unicast-igp-topology";
  prefix "l3t";
  import ietf-network {
    prefix "nw";
  }
  import ietf-network-topology {
    prefix "nt";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  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:     Jeffrey Haas
                  <mailto:jhaas@pfr.org>

    Editor:        Alexander Clemm
                  <mailto:alex@cisco.com>
```

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

Editor: Robert Varga
 <mailto:rovarga@cisco.com>

Editor: Tony Tkacik
 <mailto:ttkacik@cisco.com>

Editor: Xufeng Liu
 <mailto:xufeng.liu@ericsson.com>

Editor: Igor Bryskin
 <mailto:Igor.Bryskin@huawei.com>

Editor: Aihua Guo
 <mailto:aguo@advaoptical.com>

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

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

Editor: Vishnu Pavan Beeram
 <mailto:vbeeram@juniper.net>";

description

"This module defines a model for Layer 3 Unicast IGP
topologies.

Copyright (c) 2015 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-02;
see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to
draft-ietf-i2rs-yang-network-topo-02 with RFC
number when published (i.e. RFC xxxx).";

```

revision "2015-12-11" {
  description
    "Initial revision.
    NOTE TO RFC EDITOR: Please replace the following reference
    to draft-ietf-i2rs-yang-l3-topology-01 with
    RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-l3-topology-01";
}

typedef igp-event-type {
  type enumeration {
    enum "add" {
      value 0;
      description
        "An IGP node or link or prefix or termination-point has
        been added";
    }
    enum "remove" {
      value 1;
      description
        "An IGP node or link or prefix or termination-point has
        been removed";
    }
    enum "update" {
      value 2;
      description
        "An IGP node or link or prefix or termination-point has
        been updated";
    }
  }
  description "IGP Event type for notifications";
} // igp-event-type

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

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

```

```

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

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

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

grouping tp-ref {
  description
    "Grouping for an absolute reference to a termination point.";
  uses node-ref;
  leaf tp-ref {

```

```

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

grouping igp-prefix-attributes {
    description
        "IGP prefix attributes";
    leaf prefix {
        type inet:ip-prefix;
        description
            "IP prefix value";
    }
    leaf metric {
        type uint32;
        description
            "Prefix metric";
    }
    leaf-list flag {
        type flag-type;
        description
            "Prefix flags";
    }
}

grouping l3-unicast-igp-topology-type {
    description "Identify the topology type to be L3 unicast.";
    container l3-unicast-igp-topology {
        presence "indicates L3 Unicast IGP Topology";
        description
            "The presence of the container node indicates L3 Unicast
            IGP Topology";
    }
}

grouping igp-topology-attributes {
    description "Topology scope attributes";
    container igp-topology-attributes {
        description "Containing topology attributes";
        leaf name {
            type string;
            description

```



```

        "Name of the topology";
    }
    leaf-list flag {
        type flag-type;
        description
            "Topology flags";
    }
}

grouping igp-node-attributes {
    description "IGP node scope attributes";
    container igp-node-attributes {
        description
            "Containing node attributes";
        leaf name {
            type inet:domain-name;
            description
                "Node name";
        }
        leaf-list flag {
            type flag-type;
            description
                "Node operational flags";
        }
        leaf-list router-id {
            type inet:ip-address;
            description
                "Router-id for the node";
        }
        list prefix {
            key "prefix";
            description
                "A list of prefixes along with their attributes";
            uses igp-prefix-attributes;
        }
    }
}

grouping igp-link-attributes {
    description
        "IGP link scope attributes";
    container igp-link-attributes {
        description
            "Containing link attributes";
        leaf name {
            type string;
            description

```

```

        "Link Name";
    }
    leaf-list flag {
        type flag-type;
        description
            "Link flags";
    }
    leaf metric {
        type uint32 {
            range "0..16777215" {
                description
                    "This is a metric that can take a 3 byte metric,
                     commonly used in OSPF/ISIS";
            }
        }
        description
            "Link Metric";
    }
}
}
}

grouping igp-termination-point-attributes {
    description "IGP termination point scope attributes";
    container igp-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";
                }
            }
        }
    }
}
} // grouping igp-termination-point-attributes

augment "/nw:networks/nw:network/nw:network-types" {

```

```

    description
      "Introduce new network type for L3 unicast IGP topology";
    uses l3-unicast-igp-topology-type;
  }

  augment "/nw:networks/nw:network" {
    when "nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description
      "Configuration parameters for L3 unicast IPG for the network
      as a whole";
    uses igp-topology-attributes;
  }

  augment "/nw:networks/nw:network/nw:node" {
    when "../nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description
      "Configuration parameters for L3 unicast IPG at the node
      level";
    uses igp-node-attributes;
  }

  augment "/nw:networks/nw:network/nt:link" {
    when "../nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description
      "Augment topology link configuration";
    uses igp-link-attributes;
  }

  augment "/nw:networks/nw:network/nw:node/"
    + "nt:termination-point" {
    when "../../nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description "Augment topology termination point configuration";
  }

```

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

notification igp-node-event {
    description
        "Notification event for IGP node";
    leaf igp-event-type {
        type igp-event-type;
        description
            "Event type";
    }
    uses node-ref;
    uses l3-unicast-igp-topology-type;
    uses igp-node-attributes;
}

notification igp-link-event {
    description
        "Notification event for IGP link";
    leaf igp-event-type {
        type igp-event-type;
        description
            "Event type";
    }
    uses link-ref;
    uses l3-unicast-igp-topology-type;
    uses igp-link-attributes;
}

notification igp-prefix-event {
    description
        "Notification event for IGP prefix";
    leaf igp-event-type {
        type igp-event-type;
        description
            "Event type";
    }
    uses node-ref;
    uses l3-unicast-igp-topology-type;
    container prefix {
        description
            "Containing IPG prefix attributes";
        uses igp-prefix-attributes;
    }
}

notification termination-point-event {
    description
```

```
        "Notification event for IGP termination point";
    leaf igp-event-type {
        type igp-event-type;
        description
            "Event type";
    }
    uses tp-ref;
    uses l3-unicast-igp-topology-type;
    uses igp-termination-point-attributes;
}
}
```

<CODE ENDS>

5. OSPF Topology YANG Module

```
<CODE BEGINS> file "ietf-ospf-topology@2015-12-11.yang"
module ietf-ospf-topology {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-ospf-topology";
    prefix "ospf";

    import ietf-inet-types {
        prefix "inet";
    }
    import ietf-network {
        prefix "nw";
    }
    import ietf-network-topology {
        prefix "nt";
    }
    import ietf-l3-unicast-igp-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>

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

        WG Chair:     Jeffrey Haas
                     <mailto:jhaas@pfrc.org>
```

Editor: Alexander Clemm
 <mailto:alex@cisco.com>

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

Editor: Robert Varga
 <mailto:rovarga@cisco.com>

Editor: Tony Tkacik
 <mailto:ttkacik@cisco.com>

Editor: Xufeng Liu
 <mailto:xufeng.liu@ericsson.com>

Editor: Igor Bryskin
 <mailto:Igor.Bryskin@huawei.com>

Editor: Aihua Guo
 <mailto:aguo@advaoptical.com>

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

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

Editor: Vishnu Pavan Beeram
 <mailto:vbeeram@juniper.net>";

description

"This module defines a model for OSPF network topologies.

Copyright (c) 2015 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-02;
see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to

```

    draft-ietf-i2rs-yang-network-topo-02 with RFC
    number when published (i.e. RFC xxxx).";

revision "2015-12-11" {
  description
    "Initial revision.
    NOTE TO RFC EDITOR: Please replace the following reference
    to draft-ietf-i2rs-yang-l3-topology-01 with
    RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-l3-topology-01";
}

typedef area-id {
  type uint32;
  description
    "OSPF Area ID";
}

grouping ospf-topology-type {
  description
    "Identifies the OSPF topology type.";
  container ospf {
    presence "indiates OSPF Topology";
    description
      "Its presence identifies the OSPF topology type.";
  }
}

augment "/nw:networks/nw:network/nw:network-types/"
+ "l3t:l3-unicast-igp-topology" {
  description
    "Defines the OSPF topology type.";
  uses ospf-topology-type;
}

augment "/nw:networks/nw:network/l3t:igp-topology-attributes" {
  when "../nw:network-types/l3t:l3-unicast-igp-topology/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;
    }
  }
}

```

```

        description
            "OSPF area ID";
    }
}

augment "/nw:networks/nw:network/nw:node/l3t:igp-node-attributes" {
    when "../nw:network-types/l3t:l3-unicast-igp-topology/ospf" {
        description
            "Augment only for OSPF topology";
    }
    description
        "Augment node configuration";
    uses ospf-node-attributes;
}

augment "/nw:networks/nw:network/nt:link/l3t:igp-link-attributes" {
    when "../nw:network-types/l3t:l3-unicast-igp-topology/ospf" {
        description
            "Augment only for OSPF topology";
    }
    description
        "Augment link configuration";
    uses ospf-link-attributes;
}

augment "/nw:networks/nw:network/nw:node/" +
    "l3t:igp-node-attributes/l3t:prefix" {
    when "../nw:network-types/l3t:l3-unicast-igp-topology/"
        +"ospf" {
        description
            "Augment only for OSPF topology";
    }
    description
        "Augment prefix";
    uses ospf-prefix-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 "../router-type/pseudonode" {
        description
            "Valid only for pseudonode";
    }
    type uint32;
    default "0";
    description
        "For pseudonodes, DR interface-id";
}
leaf-list multi-topology-id {
    type uint8 {
        range "0..127";
    }
    max-elements "128";
    description
        "List of Multi-Topology Identifier up-to 128 (0-127).
        See RFC 4915";
}
leaf capabilities {

```

```

    type bits {
        bit graceful-restart-capable {
            position 0;
            description
                "Graceful restart capable";
        }
        bit graceful-restart-helper {
            position 1;
            description
                "Graceful restart helper";
        }
        bit stub-router-support {
            position 2;
            description
                "Stub router support";
        }
        bit traffic-engineering-support {
            position 3;
            description
                "Traffic engineering support";
        }
        bit point-to-point-over-lan {
            position 4;
            description
                "Support point to point over LAN";
        }
        bit experimental-te {
            position 5;
            description
                "Support experimental traffic engineering";
        }
    }
    description
        "OSPF capabilities as bit vector. RFC 4970";
}
}
}

grouping ospf-link-attributes {
    description
        "OSPF link scope attributes";
    container ospf-link-attributes {
        description
            "Containing OSPF link attributes";
        leaf multi-topology-id {
            type uint8 {
                range "0..127";
            }
        }
    }
}

```

```

        description "Muti topology ID";
    }
}
} // ospf-link-attributes

grouping ospf-prefix-attributes {
    description
        "OSPF prefix attributes";
    container ospf-prefix-attributes {
        description
            "Containing prefix attributes";
        leaf forwarding-address {
            when "../l3t:l3-unicast-igp-topology/l3t:ospf/" +
                "l3t:router-type/l3t:asbr" {
                description "Valid only for ABSR";
            }
            type inet:ipv4-address;
            description
                "Forwarding address for ABSR";
        }
    }
}

augment "/l3t:igp-node-event" {
    description
        "OSPF node event";
    uses ospf-topology-type;
    uses ospf:ospf-node-attributes;
}

augment "/l3t:igp-link-event" {
    description
        "OSPF link event";
    uses ospf-topology-type;
    uses ospf:ospf-link-attributes;
}

augment "/l3t:igp-prefix-event" {
    description
        "OSPF prefix event";
    uses ospf-topology-type;
    uses ospf:ospf-prefix-attributes;
}
}

<CODE ENDS>

```

6. ISIS Topology YANG Module

```
<CODE BEGINS> file "isis-topology@2015-12-11.yang"
module ietf-isis-topology {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-isis-topology";
  prefix "isis";

  import ietf-network {
    prefix "nw";
  }
  import ietf-network-topology {
    prefix "nt";
  }
  import ietf-l3-unicast-igp-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>

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

     WG Chair:    Jeffrey Haas
                  <mailto:jhaas@pfr.org>

     Editor:      Alexander Clemm
                  <mailto:alex@cisco.com>

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

     Editor:      Robert Varga
                  <mailto:rovarga@cisco.com>

     Editor:      Tony Tkacik
                  <mailto:ttkacik@cisco.com>

     Editor:      Xufeng Liu
                  <mailto:xufeng.liu@ericsson.com>

     Editor:      Igor Bryskin
                  <mailto:Igor.Bryskin@huawei.com>
```

Editor: Aihua Guo
<mailto:aguo@advaoptical.com>

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

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

Editor: Vishnu Pavan Beeram
<mailto:vbeeram@juniper.net>";

description

"This module defines a model for IS-IS network topologies.

Copyright (c) 2015 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-02; see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to draft-ietf-i2rs-yang-network-topo-02 with RFC number when published (i.e. RFC xxxx).";

```
revision "2015-12-11" {
  description
    "Initial revision.
    NOTE TO RFC EDITOR: Please replace the following reference
    to draft-ietf-i2rs-yang-l3-topology-01 with
    RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-l3-topology-01";
}

typedef iso-system-id {
  type string {
    pattern '[0-9a-fA-F]{4}(\.[0-9a-fA-F]{4}){2}';
  }
  description
```

```

        "ISO System ID. RFC 1237";
    }

    typedef iso-pseudonode-id {
        type string {
            pattern '[0-9a-fA-F]{2}';
        }
        description
            "ISO pseudonode id for broadcast network";
    }

    typedef iso-net-id {
        type string {
            pattern '[0-9a-fA-F]{2}((\.[0-9a-fA-F]{4}){6})';
        }
        description
            "ISO NET ID. RFC 1237";
    }

    grouping isis-topology-type {
        description
            "Identifies the ISIS topology type.";
        container isis {
            presence "Indicates ISIS Topology";
            description
                "Its presence identifies the ISIS topology type.";
        }
    }

    augment "/nw:networks/nw:network/nw:network-types/"
        + "l3t:l3-unicast-igp-topology" {
        description
            "Defines the ISIS topology type.";
        uses isis-topology-type;
    }

    augment "/nw:networks/nw:network/l3t:igp-topology-attributes" {
        when "../nw:network-types/l3t:l3-unicast-igp-topology/isis" {
            description
                "Augment only for ISIS topology";
        }
        description
            "Augment topology configuration";
        container isis-topology-attributes {
            description
                "Containing topology attributes";
            leaf net {
                type iso-net-id;
            }
        }
    }

```

```

        description
            "ISO NET ID value";
    }
}

augment "/nw:networks/nw:network/nw:node/"+
    "l3t:igp-node-attributes" {
    when "../..nw:network-types/l3t:l3-unicast-igp-topology/isis" {
        description
            "Augment only for ISIS topology";
    }
    description
        "Augment node configuration";
    uses isis-node-attributes;
}

augment "/nw:networks/nw:network/nt:link/l3t:igp-link-attributes" {
    when "../..nw:network-types/l3t:l3-unicast-igp-topology/isis" {
        description
            "Augment only for ISIS topology";
    }
    description
        "Augment link configuration";
    uses isis-link-attributes;
}

grouping isis-node-attributes {
    description
        "ISIS node scope attributes";
    container isis-node-attributes {
        description
            "Containing node attributes";
        container iso {
            description
                "Containing ISO attributes";
            leaf iso-system-id {
                type iso-system-id;
                description
                    "ISO system ID";
            }
            leaf iso-pseudonode-id {
                type iso-pseudonode-id;
                default "00";
                description
                    "Pseudonode ID";
            }
        }
    }
}

```

```

    leaf-list net {
        type iso-net-id;
        max-elements 3;
        description
            "List of ISO NET IDs";
    }
    leaf-list multi-topology-id {
        type uint8 {
            range "0..127";
        }
        max-elements "128";
        description
            "List of Multi Topology Identifier upto 128 (0-127).
            RFC 4915";
    }
    choice router-type {
        description
            "Indicates router type";
        case level-2 {
            leaf level-2 {
                type empty;
                description
                    "Level-2 only";
            }
        }
        case level-1 {
            leaf level-1 {
                type empty;
                description
                    "Level-1 only";
            }
        }
        case level-1-2 {
            leaf level-1-2 {
                type empty;
                description
                    "Level-1 and Level-2";
            }
        }
    }
}

grouping isis-link-attributes {
    description
        "ISIS link scope attributes";
    container isis-link-attributes {
        description

```



```
        "Containing link attributes";
    leaf multi-topology-id {
        type uint8 {
            range "0..127";
        }
        description
            "Multi topology ID";
    }
}

augment "/l3t:igp-node-event" {
    description
        "ISIS node event";
    uses isis-topology-type;
    uses isis-node-attributes;
}

augment "/l3t:igp-link-event" {
    description
        "ISIS link event";
    uses isis-topology-type;
    uses isis-link-attributes;
}
}
```

<CODE ENDS>

7. Security Considerations

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

8. Contributors

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

- o Ken Gray, Juniper Networks
- o Tom Nadeau, Brocade
- o Aleksandr Zhdankin, Cisco

9. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Ladislav Lhotka, Andy Bierman, Carlos Pignataro, Joel Halpern, Juergen Schoenwaelder, Alia Atlas, and Susan Hares.

10. References

10.1. Normative References

- [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-02, December 2015.
- [I-D.draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-08, October 2015.
- [I-D.draft-ietf-netmod-rfc6020bis-09]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", I-D draft-ietf-netmod-rfc6020bis-09, December 2015.
- [I-D.draft-ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", I-D draft-ietf-netmod-yang-json-06, October 2015.
- [RFC1195] Callon, R., "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments", RFC 1195, December 1990.
- [RFC2178] Moy, J., "OSPF Version 2", RFC 2178, July 1997.
- [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.
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

10.2. 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.

Authors' Addresses

Alexander Clemm
Cisco

E-Mail: alex@cisco.com

Jan Medved
Cisco

E-Mail: jmedved@cisco.com

Robert Varga
Pantheon Technologies SRO

E-Mail: robert.varga@pantheon.sk

Tony Tkacik
Pantheon Technologies SRO

E-Mail: tony.tkacik@pantheon.sk

Xufeng Liu
Ericsson

E-Mail: xufeng.liu@ericsson.com

Igor Bryskin
Huawei

E-Mail: Igor.Bryskin@huawei.com

Aihua Guo
Adva Optical

EMail: aguo@advaoptical.com

Hariharan Ananthakrishnan
Packet Design

EMail: hari@packetdesign.com

Nitin Bahadur
Bracket Computing

EMail: nitin_bahadur@yahoo.com

Vishnu Pavan Beeram
Juniper Networks

EMail: vbeeram@juniper.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 14, 2016

A. Clemm
J. Medved
Cisco
R. Varga
T. Tkacik
Pantheon Technologies SRO
X. Liu
Kuatro Technologies
I. Bryskin
Huawei
A. Guo
Adva Optical
H. Ananthakrishnan
Packet Design
N. Bahadur
Bracket Computing
V. Beeram
Juniper Networks
June 12, 2016

A YANG Data Model for Layer 3 Topologies
draft-ietf-i2rs-yang-l3-topology-02.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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on December 14, 2016.

Copyright Notice

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

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Definitions and Acronyms	4
3. Model overview	5
3.1. Model structure	5
3.2. Layer 3 Unicast - IGP	6
3.3. OSPF Topology	7
3.4. IS-IS Topology	9
4. Layer 3 Unicast IGP Topology YANG Module	10
5. OSPF Topology YANG Module	19
6. ISIS Topology YANG Module	26
7. Security Considerations	31
8. Contributors	31
9. Acknowledgements	32
10. References	32
10.1. Normative References	32
10.2. Informative References	33
Authors' Addresses	33

1. Introduction

This document introduces a YANG [RFC6020] [RFC6991] [I-D.draft-ietf-netmod-rfc6020bis] data model for Layer 3 network topologies. 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]. An earlier revision of that Internet Draft contained not just the general model for network topologies, but also the model for layer 3 network topologies that is being specified here. However, we decided to "split" the earlier draft to separate the truly general aspects of a topology data model, which apply to any type of topology, from the application of this model to a particular domain, here: a Layer 3 network.

Specific topology types that are covered in this document include Layer 3 Unicast IGP, IS-IS [RFC1195], and OSPF [RFC2178]. In addition, this documents defines a set of traffic engineering extensions.

There are multiple applications for such a data model and 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.

There are several reasons to choose YANG to define the data model. Data defined using YANG can be exposed by a server to client applications and controllers via Netconf [RFC6241] or via a ReST-like Interface [I-D.draft-ietf-netconf-restconf] [I-D.draft-ietf-netmod-yang-json]. The fact that it can be used with different protocols and interfaces provides for a degree of "future-proofing" of model implementations. Also, YANG can serve as the basis for model-driven toolchains, such as used in the Open Daylight project.

The data model is defined in several YANG modules:

- o Module "ietf-l3-unicast-igp-topology" defines a model for Layer 3 Unicast IGP topologies. To do so, it augments general network topology model defined in [I-D.draft-ietf-i2rs-yang-network-topol] with information specific to Layer 3 Unicast IGP. In doing so, it also illustrates the extension patterns associated with extending respectively augmenting the general topology model to meet the needs of a specific topology.
- o Module "ietf-ospf-topology" defines a topology model for OSPF, building on and extending the Layer 3 Unicast IGP topology model. It serves as an example of how the general topology model can be refined across multiple levels.
- o Module "ietf-isis-topology" defines a topology model for IS-IS, again building on and extending the Layer 3 Unicast IGP topology model.

Information that is kept in the Traffic Engineering Database (TED) is specified in a separate model and outside the scope of this specification.

2. Definitions and Acronyms

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

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

HTTP: Hyper-Text Transfer Protocol

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

LSP: Label Switched Path

NETCONF: Network Configuration Protocol

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

URI: Uniform Resource Identifier

ReST: Representational State Transfer, a style of stateless interface and protocol that is generally carried over HTTP

SRLG: Shared Risk Link Group

TED: Traffic Engineering Database

YANG: A data definition language for NETCONF

3. Model overview

This section provides an overview of the Layer 3 network topology model.

3.1. Model structure

The network topology model is defined by the following YANG modules, whose relationship is roughly depicted in the figure below. The base network topology is included in the diagram for completeness.

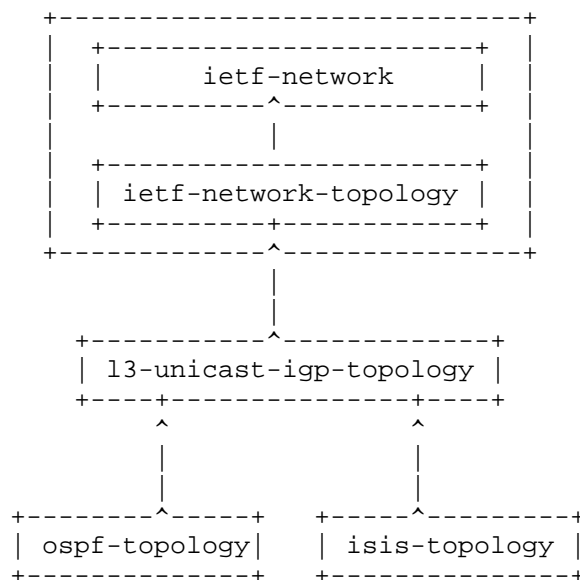


Figure 1: Overall model structure

YANG modules `ietf-network` and `ietf-network-topology` collectively define the basic network topology model. YANG module `ietf-l3-unicast-igp-topology` augments those models with additional definitions needed to represent Layer 3 Unicast IGP topologies. This module in turn is augmented by YANG modules with additional definitions for OSPF and for IS-IS topologies, `ietf-ospf-topology` and `ietf-isis-topology`, respectively.

3.2. Layer 3 Unicast - IGP

The Layer 3 Unicast IGP topology model is defined by YANG module "ietf-l3-unicast-igp-topology". The model is depicted in the following diagram. Brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parantheses enclose choice and case nodes. Notifications are not depicted. The prefix "nt:" refers to the YANG module for network topology.

```

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

```

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

- o A new network topology type is introduced, l3-unicast-igp-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 an IGP name, 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 that are associated with specific IGP topologies, 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 an IGP 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 or identifier.

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.

3.3. OSPF Topology

OSPF is the next type of topology represented in the model. OSPF represents a particular type of Layer 3 Unicast IGP. Accordingly, this time the Layer 3 Unicast IGP topology model needs to be extended. The corresponding extensions are introduced in a separate YANG module "ietf-ospf-topology", whose structure is depicted in the following diagram. For the most part, this module augments "ietf-l3-unicast-igp-topology". Like before, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses enclose choice and case nodes. A "+" at the end of a line indicates a line break. Notifications respectively augmentations of notifications are not depicted.

```

module: ietf-ospf-topology
augment /nw:networks/nw:network/nw:network-types/+
|   l3t:l3-unicast-igp-topology:
  +--rw ospf!
augment /nw:networks/nw:network/l3t:igp-topology-attributes:
  +--rw ospf-topology-attributes
    +--rw area-id?   area-id
augment /nw:networks/nw:network/nw:node/l3t:igp-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
    +--rw multi-topology-id*  uint8
    +--rw capabilities?      bits
augment /nw:networks/nw:network/nt:link/l3t:igp-link-attributes:
  +--rw ospf-link-attributes
    +--rw multi-topology-id?  uint8
augment /nw:networks/nw:network/nw:node/l3t:igp-node-attributes/+
|   l3t:prefix:
  +--rw ospf-prefix-attributes
    +--rw forwarding-address?  inet:ipv4-address

```

The module augments "ietf-l3-unicast-igp-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 igp-topology-attributes of the ietf-l3-unicast-igp-topology module. The attributes include an OSPF area-id identifying the OSPF area.
- o Additional data objects for nodes are introduced by augmenting the igp-node-attributes of the l3-unicast-igp-topology module. New objects include router-type, dr-interface-id for pseudonodes, list of multi-topology-ids, ospf node capabilities, and traffic engineering attributes.
- o Links are augmented with a multi-topology-id and traffic engineering link attributes.

- o Prefixes are augmented with OSPF specific forwarding address.

In addition, the module extends IGP node, link and prefix notifications with OSPF attributes.

3.4. IS-IS Topology

IS-IS is another type of Layer 3 Unicast IGP. Like OSPF topology, IS-IS topology is defined in a separate module, "ietf-isis-topology", which augments "ietf-l3-unicast-igp-topology". The structure is depicted in the following diagram. Like before, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses enclose choice and case nodes. A "+" at the end of a line indicates a line break. Notifications are not depicted.

```

module: ietf-isis-topology
augment /nw:networks/nw:network/nw:network-types/+
|   l3t:l3-unicast-igp-topology:
+--rw isis!
augment /nw:networks/nw:network/l3t:igp-topology-attributes:
+--rw isis-topology-attributes
+--rw net?   iso-net-id
augment /nw:networks/nw:network/nw:node/l3t:igp-node-attributes:
+--rw isis-node-attributes
+--rw iso
|   +--rw iso-system-id?       iso-system-id
|   +--rw iso-pseudonode-id?   iso-pseudonode-id
+--rw net*                     iso-net-id
+--rw multi-topology-id*       uint8
+--rw (router-type)?
+--:(level-2)
|   +--rw level-2?             empty
+--:(level-1)
|   +--rw level-1?             empty
+--:(level-1-2)
|   +--rw level-1-2?           empty
augment /nw:networks/nw:network/nt:link/l3t:igp-link-attributes:
+--rw isis-link-attributes
+--rw multi-topology-id?       uint8

```

The module augments the ietf-l3-unicast-igp-topology as follows:

- o A new topology type is introduced for isis.
- o Additional topology attributes are introduced in a new grouping which augments "igp-topology-attributes" of the ietf-l3-unicast-

igp-topology module. The attributes include an ISIS NET-id identifying the area.

- o Additional data objects for nodes are introduced by augmenting "igp-node-attributes" of the ietf-l3-unicast-igp-topology module. New objects include router-type, iso-system-id to identify the router, a list of multi-topology-id, a list of NET ids, and traffic engineering attributes.
- o Links are augmented with multi-topology-id and traffic engineering link attributes.

In addition, the module augments IGP nodes and links with ISIS attributes.

4. Layer 3 Unicast IGP Topology YANG Module

```
<CODE BEGINS> file "ietf-l3-unicast-igp-topology@2016-06-12.yang"
module ietf-l3-unicast-igp-topology {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-l3-unicast-igp-topology";
  prefix "l3t";
  import ietf-network {
    prefix "nw";
  }
  import ietf-network-topology {
    prefix "nt";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  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:      Alexander Clemm
                  <mailto:alex@cisco.com>
```

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

Editor: Robert Varga
<mailto:rovarga@cisco.com>

Editor: Tony Tkacik
<mailto:ttkacik@cisco.com>

Editor: Xufeng Liu
<mailto:xliu@kuatrotech.com>

Editor: Igor Bryskin
<mailto:Igor.Bryskin@huawei.com>

Editor: Aihua Guo
<mailto:aguo@advaoptical.com>

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

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

Editor: Vishnu Pavan Beeram
<mailto:vbeeram@juniper.net>";

description

"This module defines a model for Layer 3 Unicast IGP topologies.

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

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

This version of this YANG module is part of draft-ietf-i2rs-yang-network-topo-02; see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to draft-ietf-i2rs-yang-network-topo-02 with RFC number when published (i.e. RFC xxxx).";

```
revision "2016-06-12" {
  description
    "Initial revision.
    NOTE TO RFC EDITOR: Please replace the following reference
    to draft-ietf-i2rs-yang-l3-topology-01 with
    RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-l3-topology-02";
}

typedef igp-event-type {
  type enumeration {
    enum "add" {
      value 0;
      description
        "An IGP node or link or prefix or termination-point has
        been added";
    }
    enum "remove" {
      value 1;
      description
        "An IGP node or link or prefix or termination-point has
        been removed";
    }
    enum "update" {
      value 2;
      description
        "An IGP node or link or prefix or termination-point has
        been updated";
    }
  }
  description "IGP Event type for notifications";
} // igp-event-type

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

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



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

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

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

grouping tp-ref {
  description
    "Grouping for an absolute reference to a termination point.";
  uses node-ref;
  leaf tp-ref {
```

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

grouping igp-prefix-attributes {
  description
    "IGP prefix attributes";
  leaf prefix {
    type inet:ip-prefix;
    description
      "IP prefix value";
  }
  leaf metric {
    type uint32;
    description
      "Prefix metric";
  }
  leaf-list flag {
    type flag-type;
    description
      "Prefix flags";
  }
}

grouping l3-unicast-igp-topology-type {
  description "Identify the topology type to be L3 unicast.";
  container l3-unicast-igp-topology {
    presence "indicates L3 Unicast IGP Topology";
    description
      "The presence of the container node indicates L3 Unicast
      IGP Topology";
  }
}

grouping igp-topology-attributes {
  description "Topology scope attributes";
  container igp-topology-attributes {
    description "Containing topology attributes";
    leaf name {
      type string;
      description

```

```
        "Name of the topology";
    }
    leaf-list flag {
        type flag-type;
        description
            "Topology flags";
    }
}

grouping igp-node-attributes {
    description "IGP node scope attributes";
    container igp-node-attributes {
        description
            "Containing node attributes";
        leaf name {
            type inet:domain-name;
            description
                "Node name";
        }
        leaf-list flag {
            type flag-type;
            description
                "Node operational flags";
        }
        leaf-list router-id {
            type inet:ip-address;
            description
                "Router-id for the node";
        }
        list prefix {
            key "prefix";
            description
                "A list of prefixes along with their attributes";
            uses igp-prefix-attributes;
        }
    }
}

grouping igp-link-attributes {
    description
        "IGP link scope attributes";
    container igp-link-attributes {
        description
            "Containing link attributes";
        leaf name {
            type string;
            description
```

```
        "Link Name";
    }
    leaf-list flag {
        type flag-type;
        description
            "Link flags";
    }
    leaf metric {
        type uint32 {
            range "0..16777215" {
                description
                    "This is a metric that can take a 3 byte metric,
                     commonly used in OSPF/ISIS";
            }
        }
        description
            "Link Metric";
    }
}

grouping igp-termination-point-attributes {
    description "IGP termination point scope attributes";
    container igp-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";
                }
            }
        }
    }
}
} // grouping igp-termination-point-attributes

augment "/nw:networks/nw:network/nw:network-types" {
```

```
    description
      "Introduce new network type for L3 unicast IGP topology";
    uses l3-unicast-igp-topology-type;
  }

  augment "/nw:networks/nw:network" {
    when "nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description
      "Configuration parameters for L3 unicast IPG for the network
      as a whole";
    uses igp-topology-attributes;
  }

  augment "/nw:networks/nw:network/nw:node" {
    when "../nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description
      "Configuration parameters for L3 unicast IPG at the node
      level";
    uses igp-node-attributes;
  }

  augment "/nw:networks/nw:network/nt:link" {
    when "../nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description
      "Augment topology link configuration";
    uses igp-link-attributes;
  }

  augment "/nw:networks/nw:network/nw:node/"
    + "nt:termination-point" {
    when "../../nw:network-types/l3-unicast-igp-topology" {
      description
        "Augmentation parameters apply only for networks with
        L3 unicast IGP topology";
    }
    description "Augment topology termination point configuration";
  }
```

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

  notification igp-node-event {
    description
      "Notification event for IGP node";
    leaf igp-event-type {
      type igp-event-type;
      description
        "Event type";
    }
    uses node-ref;
    uses l3-unicast-igp-topology-type;
    uses igp-node-attributes;
  }

  notification igp-link-event {
    description
      "Notification event for IGP link";
    leaf igp-event-type {
      type igp-event-type;
      description
        "Event type";
    }
    uses link-ref;
    uses l3-unicast-igp-topology-type;
    uses igp-link-attributes;
  }

  notification igp-prefix-event {
    description
      "Notification event for IGP prefix";
    leaf igp-event-type {
      type igp-event-type;
      description
        "Event type";
    }
    uses node-ref;
    uses l3-unicast-igp-topology-type;
    container prefix {
      description
        "Containing IPG prefix attributes";
      uses igp-prefix-attributes;
    }
  }

  notification termination-point-event {
    description
```

```
        "Notification event for IGP termination point";
    leaf igp-event-type {
        type igp-event-type;
        description
            "Event type";
    }
    uses tp-ref;
    uses l3-unicast-igp-topology-type;
    uses igp-termination-point-attributes;
}
}
<CODE ENDS>
```

5. OSPF Topology YANG Module

```
<CODE BEGINS> file "ietf-ospf-topology@2016-06-12.yang"
module ietf-ospf-topology {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-ospf-topology";
    prefix "ospf";

    import ietf-inet-types {
        prefix "inet";
    }
    import ietf-network {
        prefix "nw";
    }
    import ietf-network-topology {
        prefix "nt";
    }
    import ietf-l3-unicast-igp-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>

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

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

        Editor:       Alexander Clemm
```

<mailto:alex@cisco.com>

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

Editor: Robert Varga
<mailto:rovarga@cisco.com>

Editor: Tony Tkacik
<mailto:ttkacik@cisco.com>

Editor: Xufeng Liu
<mailto:xliu@kuatrotech.com>

Editor: Igor Bryskin
<mailto:Igor.Bryskin@huawei.com>

Editor: Aihua Guo
<mailto:aguo@advaoptical.com>

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

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

Editor: Vishnu Pavan Beeram
<mailto:vbeeram@juniper.net>";

description

"This module defines a model for OSPF network topologies.

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

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

This version of this YANG module is part of draft-ietf-i2rs-yang-network-topo-02; see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to draft-ietf-i2rs-yang-network-topo-02 with RFC


```
    number when published (i.e. RFC xxxx).";

revision "2016-06-12" {
  description
    "Initial revision.
     NOTE TO RFC EDITOR: Please replace the following reference
     to draft-ietf-i2rs-yang-l3-topology-02 with
     RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-l3-topology-02";
}

typedef area-id {
  type uint32;
  description
    "OSPF Area ID";
}

grouping ospf-topology-type {
  description
    "Identifies the OSPF topology type.";
  container ospf {
    presence "indiates OSPF Topology";
    description
      "Its presence identifies the OSPF topology type.";
  }
}

augment "/nw:networks/nw:network/nw:network-types/"
+ "l3t:l3-unicast-igp-topology" {
  description
    "Defines the OSPF topology type.";
  uses ospf-topology-type;
}

augment "/nw:networks/nw:network/l3t:igp-topology-attributes" {
  when "../nw:network-types/l3t:l3-unicast-igp-topology/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;
      description

```

```
        "OSPF area ID";
    }
}

augment "/nw:networks/nw:network/nw:node/l3t:igp-node-attributes" {
    when "../../../nw:network-types/l3t:l3-unicast-igp-topology/ospf" {
        description
            "Augment only for OSPF topology";
    }
    description
        "Augment node configuration";
    uses ospf-node-attributes;
}

augment "/nw:networks/nw:network/nt:link/l3t:igp-link-attributes" {
    when "../../../nw:network-types/l3t:l3-unicast-igp-topology/ospf" {
        description
            "Augment only for OSPF topology";
    }
    description
        "Augment link configuration";
    uses ospf-link-attributes;
}

augment "/nw:networks/nw:network/nw:node/" +
    "l3t:igp-node-attributes/l3t:prefix" {
    when "../../../nw:network-types/l3t:l3-unicast-igp-topology/"
        + "ospf" {
        description
            "Augment only for OSPF topology";
    }
    description
        "Augment prefix";
    uses ospf-prefix-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 "../router-type/pseudonode" {
        description
            "Valid only for pseudonode";
    }
    type uint32;
    default "0";
    description
        "For pseudonodes, DR interface-id";
}
leaf-list multi-topology-id {
    type uint8 {
        range "0..127";
    }
    max-elements "128";
    description
        "List of Multi-Topology Identifier up-to 128 (0-127).
        See RFC 4915";
}
leaf capabilities {
    type bits {
```

```
    bit graceful-restart-capable {
        position 0;
        description
            "Graceful restart capable";
    }
    bit graceful-restart-helper {
        position 1;
        description
            "Graceful restart helper";
    }
    bit stub-router-support {
        position 2;
        description
            "Stub router support";
    }
    bit traffic-engineering-support {
        position 3;
        description
            "Traffic engineering support";
    }
    bit point-to-point-over-lan {
        position 4;
        description
            "Support point to point over LAN";
    }
    bit experimental-te {
        position 5;
        description
            "Support experimental traffic engineering";
    }
}
description
    "OSPF capabilities as bit vector. RFC 4970";
}
}
}

grouping ospf-link-attributes {
    description
        "OSPF link scope attributes";
    container ospf-link-attributes {
        description
            "Containing OSPF link attributes";
        leaf multi-topology-id {
            type uint8 {
                range "0..127";
            }
            description "Muti topology ID";
        }
    }
}
```

```
    }
  }
} // ospf-link-attributes

grouping ospf-prefix-attributes {
  description
    "OSPF prefix attributes";
  container ospf-prefix-attributes {
    description
      "Containing prefix attributes";
    leaf forwarding-address {
      when "../l3t:l3-unicast-igp-topology/l3t:ospf/" +
        "l3t:router-type/l3t:asbr" {
        description "Valid only for ABSR";
      }
      type inet:ipv4-address;
      description
        "Forwarding address for ABSR";
    }
  }
}

augment "/l3t:igp-node-event" {
  description
    "OSPF node event";
  uses ospf-topology-type;
  uses ospf:ospf-node-attributes;
}

augment "/l3t:igp-link-event" {
  description
    "OSPF link event";
  uses ospf-topology-type;
  uses ospf:ospf-link-attributes;
}

augment "/l3t:igp-prefix-event" {
  description
    "OSPF prefix event";
  uses ospf-topology-type;
  uses ospf:ospf-prefix-attributes;
}
}

<CODE ENDS>
```

6. ISIS Topology YANG Module

```
<CODE BEGINS> file "ietf-isis-topology@2016-06-12.yang"
module ietf-isis-topology {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-isis-topology";
  prefix "isis";

  import ietf-network {
    prefix "nw";
  }
  import ietf-network-topology {
    prefix "nt";
  }
  import ietf-l3-unicast-igp-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>

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

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

     Editor:      Alexander Clemm
                  <mailto:alex@cisco.com>

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

     Editor:      Robert Varga
                  <mailto:rovarga@cisco.com>

     Editor:      Tony Tkacik
                  <mailto:ttkacik@cisco.com>

     Editor:      Xufeng Liu
                  <mailto:xliu@kuatrotech.com>

     Editor:      Igor Bryskin
                  <mailto:Igor.Bryskin@huawei.com>
```

Editor: Aihua Guo
<mailto:aguo@advaoptical.com>

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

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

Editor: Vishnu Pavan Beeram
<mailto:vbeeram@juniper.net>";

description

"This module defines a model for IS-IS network topologies.

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

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

This version of this YANG module is part of draft-ietf-i2rs-yang-network-topo-02; see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to draft-ietf-i2rs-yang-network-topo-02 with RFC number when published (i.e. RFC xxxx).";

```
revision "2016-06-12" {
  description
    "Initial revision.
    NOTE TO RFC EDITOR: Please replace the following reference
    to draft-ietf-i2rs-yang-l3-topology-02 with
    RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-l3-topology-02";
}

typedef iso-system-id {
  type string {
    pattern '[0-9a-fA-F]{4}(\.[0-9a-fA-F]{4}){2}';
  }
  description
```

```
    "ISO System ID. RFC 1237";
}

typedef iso-pseudonode-id {
  type string {
    pattern '[0-9a-fA-F]{2}';
  }
  description
    "ISO pseudonode id for broadcast network";
}

typedef iso-net-id {
  type string {
    pattern '[0-9a-fA-F]{2}((\.[0-9a-fA-F]{4}){6})';
  }
  description
    "ISO NET ID. RFC 1237";
}

grouping isis-topology-type {
  description
    "Identifies the ISIS topology type.";
  container isis {
    presence "Indicates ISIS Topology";
    description
      "Its presence identifies the ISIS topology type.";
  }
}

augment "/nw:networks/nw:network/nw:network-types/"
+ "l3t:l3-unicast-igp-topology" {
  description
    "Defines the ISIS topology type.";
  uses isis-topology-type;
}

augment "/nw:networks/nw:network/l3t:igp-topology-attributes" {
  when "../nw:network-types/l3t:l3-unicast-igp-topology/isis" {
    description
      "Augment only for ISIS topology";
  }
  description
    "Augment topology configuration";
  container isis-topology-attributes {
    description
      "Containing topology attributes";
    leaf net {
      type iso-net-id;
    }
  }
}
```



```
        description
            "ISO NET ID value";
    }
}

augment "/nw:networks/nw:network/nw:node/"+
    "l3t:igp-node-attributes" {
    when "../nw:network-types/l3t:l3-unicast-igp-topology/isis" {
        description
            "Augment only for ISIS topology";
    }
    description
        "Augment node configuration";
    uses isis-node-attributes;
}

augment "/nw:networks/nw:network/nt:link/l3t:igp-link-attributes" {
    when "../nw:network-types/l3t:l3-unicast-igp-topology/isis" {
        description
            "Augment only for ISIS topology";
    }
    description
        "Augment link configuration";
    uses isis-link-attributes;
}

grouping isis-node-attributes {
    description
        "ISIS node scope attributes";
    container isis-node-attributes {
        description
            "Containing node attributes";
        container iso {
            description
                "Containing ISO attributes";
            leaf iso-system-id {
                type iso-system-id;
                description
                    "ISO system ID";
            }
            leaf iso-pseudonode-id {
                type iso-pseudonode-id;
                default "00";
                description
                    "Pseudonode ID";
            }
        }
    }
}
```

```
leaf-list net {
  type iso-net-id;
  max-elements 3;
  description
    "List of ISO NET IDs";
}
leaf-list multi-topology-id {
  type uint8 {
    range "0..127";
  }
  max-elements "128";
  description
    "List of Multi Topology Identifier upto 128 (0-127).
    RFC 4915";
}
choice router-type {
  description
    "Indicates router type";
  case level-2 {
    leaf level-2 {
      type empty;
      description
        "Level-2 only";
    }
  }
  case level-1 {
    leaf level-1 {
      type empty;
      description
        "Level-1 only";
    }
  }
  case level-1-2 {
    leaf level-1-2 {
      type empty;
      description
        "Level-1 and Level-2";
    }
  }
}
}
}

grouping isis-link-attributes {
  description
    "ISIS link scope attributes";
  container isis-link-attributes {
    description
```

```
        "Containing link attributes";
    leaf multi-topology-id {
        type uint8 {
            range "0..127";
        }
        description
            "Multi topology ID";
    }
}

augment "/l3t:igp-node-event" {
    description
        "ISIS node event";
    uses isis-topology-type;
    uses isis-node-attributes;
}

augment "/l3t:igp-link-event" {
    description
        "ISIS link event";
    uses isis-topology-type;
    uses isis-link-attributes;
}
}
<CODE ENDS>
```

7. Security Considerations

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

8. Contributors

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

- o Ken Gray, Juniper Networks
- o Tom Nadeau, Brocade
- o Aleksandr Zhdankin, Cisco

9. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Ladislav Lhotka, Andy Bierman, Carlos Pignataro, Joel Halpern, Juergen Schoenwaelder, Alia Atlas, and Susan Hares.

10. References

10.1. Normative References

- [I-D.draft-ietf-i2rs-yang-network-topo]
Clemm, A., Medved, J., Tkacik, T., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A YANG Data Model for Network Topologies", I-D draft-ietf-i2rs-yang-network-topo-03, June 2016.
- [I-D.draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-13, April 2016.
- [I-D.draft-ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", I-D draft-ietf-netmod-rfc6020bis-13, June 2016.
- [I-D.draft-ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", I-D draft-ietf-netmod-yang-json-10, March 2016.
- [RFC1195] Callon, R., "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments", RFC 1195, December 1990.
- [RFC2178] Moy, J., "OSPF Version 2", RFC 2178, July 1997.
- [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.
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

10.2. 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-02, March 2016.

Authors' Addresses

Alexander Clemm
Cisco

EMail: alex@cisco.com

Jan Medved
Cisco

EMail: jmedved@cisco.com

Robert Varga
Pantheon Technologies SRO

EMail: robert.varga@pantheon.sk

Tony Tkacik
Pantheon Technologies SRO

EMail: tony.tkacik@pantheon.sk

Xufeng Liu
Kuatro Technologies

EMail: xliu@kuatrotech.com

Igor Bryskin
Huawei

EMail: Igor.Bryskin@huawei.com

Aihua Guo
Adva Optical

EMail: aguo@advaoptical.com

Hariharan Ananthakrishnan
Packet Design

EMail: hari@packetdesign.com

Nitin Bahadur
Bracket Computing

EMail: nitin_bahadur@yahoo.com

Vishnu Pavan Beeram
Juniper Networks

EMail: vbeeram@juniper.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 10, 2016

A. Clemm
J. Medved
R. Varga
T. Tkacik
Cisco
N. Bahadur
Bracket Computing
H. Ananthakrishnan
Packet Design
December 8, 2015

A Data Model for Network Topologies
draft-ietf-i2rs-yang-network-topo-02.txt

Abstract

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

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 June 10, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Definitions and Acronyms	7
3. Model Structure Details	7
3.1. Base Network Model	7
3.2. Base Network Topology Model	10
3.3. Extending the model	11
3.4. Discussion and selected design decisions	12
3.4.1. Container structure	12
3.4.2. Underlay hierarchies and mappings	12
3.4.3. Dealing with changes in underlay networks	13
3.4.4. Use of groupings	13
3.4.5. Cardinality and directionality of links	13
3.4.6. Multihoming and link aggregation	14
3.4.7. Mapping redundancy	14
3.4.8. Typing	14
3.4.9. Representing the same device in multiple networks	14
3.5. Open Issues	15
3.5.1. Supporting client as well as server provided network topology	16
3.5.2. Identifiers of string or URI type	16
4. YANG Modules	17
4.1. Defining the Abstract Network: network.yang	17
4.2. Creating Abstract Network Topology: network-topology.yang	21
5. Security Considerations	28
6. Contributors	28
7. Acknowledgements	28
8. References	28
8.1. Normative References	28

8.2. Informative References	29
Authors' Addresses	29

1. Introduction

This document introduces an abstract (base) YANG [RFC6020] [RFC6021] data model to represent networks and topologies. The data model is divided into two parts. The first part of the model defines a network model that allows to define network hierarchies (i.e. network stacks) and to maintain an inventory of nodes contained in a network. The second part of the model augments the basic network 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 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 model, the separation facilitates integration of network topology and network inventory models, by allowing to augment network inventory information separately and without concern for topology into the network model.

The model can be augmented to describe specifics of particular types of networks and topologies. For example, an augmenting model can provide network node information with attributes that are specific to a particular network type. Examples of augmenting models include 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 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 models allow applications to operate on an inventory or topology of any network at a generic level, where specifics of particular inventory/topology types are not required. At the same time, where data specific to a network type does come into play and the model is augmented, the instantiated data still adheres to the same structure and is represented in 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 "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 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 model, the inventory models and the topology models is shown in the following figure (dotted lines in the figure denote possible augmentations to models defined in this document).

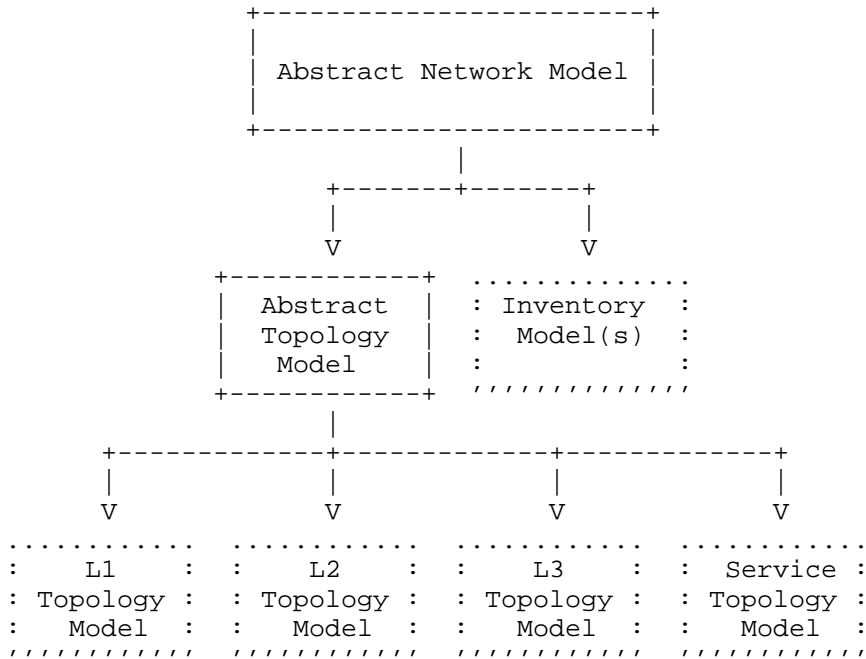


Figure 1: The network model structure

The network-topology YANG module introduced in this document, entitled "network-topology.yang", defines a generic topology 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 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 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.

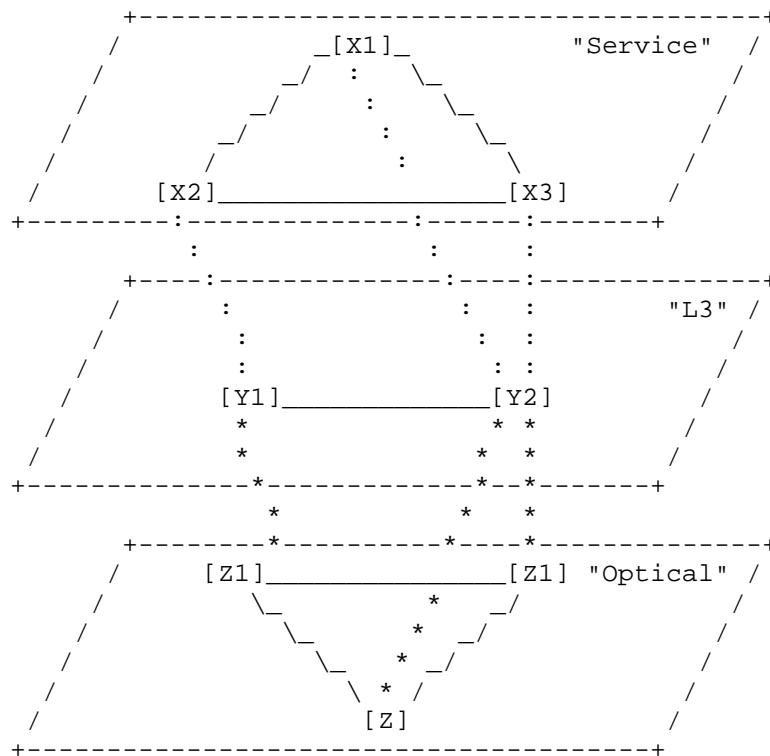


Figure 2: Topology hierarchy (stack) example

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 X2 - mapping onto a single L3 network element; 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 mapped onto multiple "Optical" network elements. This could happen, for example, if a single IP router attaches to multiple ROADMs in the optical domain.

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

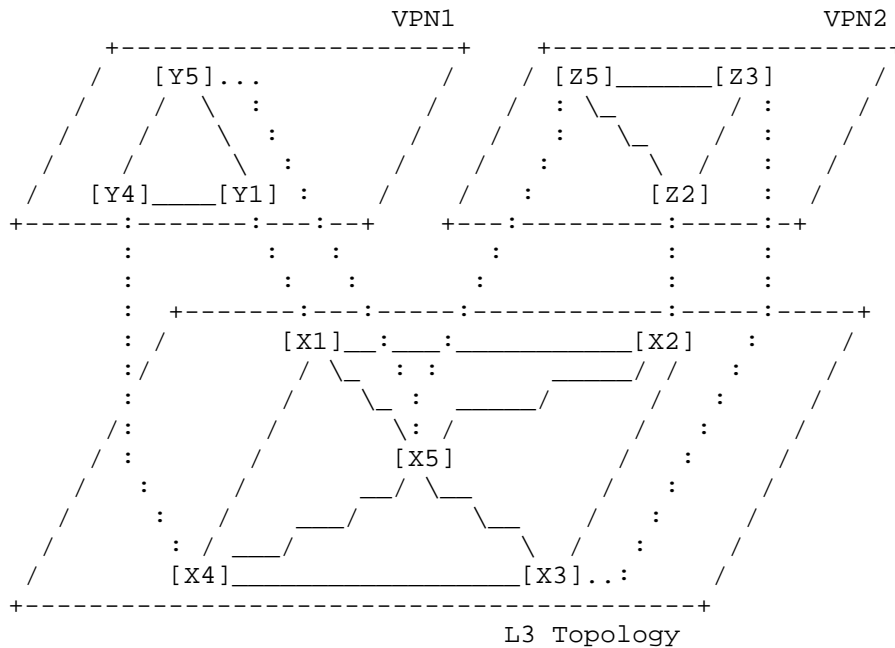


Figure 3: Topology hierarchy (stack) example

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 [topology-use-cases].

2. Definitions and Acronyms

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

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

HTTP: Hyper-Text Transfer Protocol

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

NETCONF: Network Configuration Protocol

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

URI: Uniform Resource Identifier

ReST: Representational State Transfer, a style of stateless interface and protocol that is generally carried over HTTP

YANG: A data definition language for NETCONF

3. Model Structure Details

3.1. Base Network Model

The abstract (base) network model is defined in the `network.yang` module. Its structure is shown in the following figure. Brackets enclose list keys, "rw" means configuration data, "ro" means operational state data, and "?" designates optional nodes. A "+" indicates a line break.

```

module: ietf-network
  +--rw networks
  |   +--rw network* [network-id]
  |   |   +--rw network-types
  |   |   +--rw network-id          network-id
  |   |   +--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
  |   +--ro networks-state
  |   |   +--ro network* [network-ref]
  |   |   +--ro network-ref    -> /networks/network/network-id
  |   +--ro server-provided?  boolean

```

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

The model contains a container with a list of networks, and a container with a list of network state.

Container "networks" contains 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 to represent 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 container "l3-unicast-igp-network", which in turn would contain a container "ospf-network".

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 entity takes part in multiple networks, or at multiple layers of a networking stack, the same 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 is a 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 model allows to define more than one underlay network (and node), allowing for simultaneous representation of layered network- and service topologies and 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 to represent 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.

Container "networks-state" contains data about the network state. It contains a list of networks, mirroring the corresponding list under the "networks" container and containing in each data node state information for the corresponding network. Currently, each list element contains a single state, "server-provided". This state indicates how the network came into being. If false, the network was configured by a client application, for example in the case of an overlay network that is configured by a controller application. If true, the network was populated by the server itself, respectively an application on the server that is able to discover the network.

Network data 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 populated by the server, in case of networks that can be discovered. Client applications SHOULD NOT

modify configurations of networks for which "server-provided" is true. (Please note this is an open issue for further discussion, see section Section 3.5.)

3.2. Base Network Topology Model

The abstract (base) network topology model is defined in the "network-topology.yang" module. It builds on the network model defined in the "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. Brackets enclose list keys, "rw" means configuration data, "ro" means operational state data, and "?" designates optional nodes. A "+" indicates a line break.

```

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

```

Figure 5: The structure of the abstract (base) network topology 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".

3.3. Extending the model

In order to derive a model for a specific type of network, the base 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 network.yang module. First, a new network type needs to be defined. For this, a presence container that resembles 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 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 augment modules, with the module of a network "sub-type" augmenting the module of a network "super-type".

3.4. Discussion and selected design decisions

3.4.1. Container structure

Rather than maintaining lists in separate containers, the 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 specifiers 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.

3.4.2. 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 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.

3.4.3. 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 model makes use of the "require-instance" construct of YANG 1.1 [I.D.draft-ietf-netmod-rfc6020bis].

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. 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].

3.4.4. Use of groupings

The model makes use of groupings, instead of simply defining data nodes "in-line". This allows to more easily 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.

3.4.5. Cardinality and directionality of links

The topology 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 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.

3.4.6. 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.

3.4.7. 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 model does provide for the option to configure this information explicitly. The model includes integrity constraints to allow for validating for consistency.

3.4.8. 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.

3.4.9. 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 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.

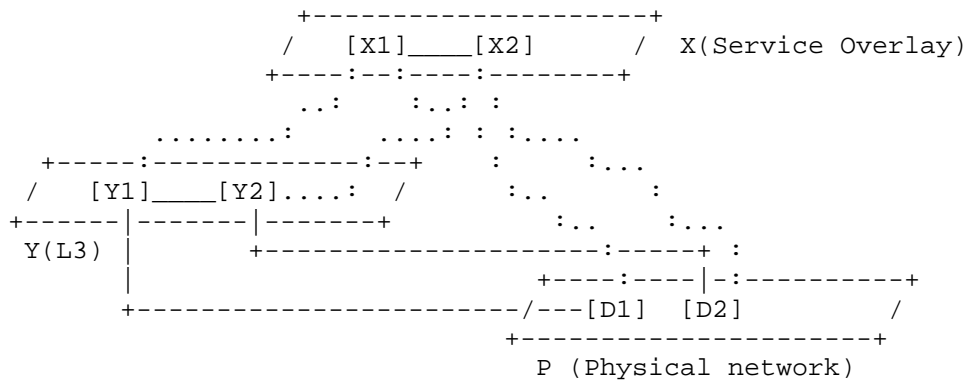


Figure 6: Topology hierarchy example - multiple underlays

In the case of a physical network, nodes represent physical devices and termination points physical ports. It should be noted that it is also conceivable to augment the 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.

3.5. Open Issues

3.5.1. Supporting client as well as server provided network topology

YANG requires data needs to be designated as either configuration or operational data, but not both, yet it is important to have all network information, including vertical cross-network dependencies, captured in one coherent 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 model. That said, it is conceivable that certain types of topology need to also be configurable by an application.

There are several alternatives in which this can be addressed. The alternative chosen in this draft does not restrict network topology information as read-only, but includes a state that indicates for each network whether it is populated by the server or by a client application. (The drawback of this solution is that it stretches its use of the configuration concept. Configuration information is subject to backup and restore, which is not applicable to server-provided information. Perhaps more serious is the ability of a client to lock a configuration and thus prevent changes to server-provided network topology while the lock is in effect. Preventing this requires special treatment of network topology related configuration information.)

In another alternative, all information about network topology that is in effect is represented as network state, i.e. as read-only information, regardless of how it came into being. For cases where network topology needs to be configured, a second branch for configurable topology information is introduced. Any network topology configuration is mirrored by network state information. A configurable network will thus be represented twice: once in the read-only list of all networks, a second time in a configuration sandbox. (The drawback of this solution is slightly increased complexity of augmentations due to two target branches.

3.5.2. Identifiers of string or URI type

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

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 (a string is a string is a string), 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.

4. YANG Modules

4.1. Defining the Abstract Network: network.yang

```
<CODE BEGINS> file "ietf-network@2015-12-08.yang"
module ietf-network {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network";
  prefix nd;

  import ietf-inet-types {
    prefix inet;
  }

  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:    Jeffrey Haas
                  <mailto:jhaas@pfr.org>

     Editor:      Alexander Clemm
                  <mailto:alex@cisco.com>

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

     Editor:      Robert Varga
                  <mailto:rovarga@cisco.com>

     Editor:      Tony Tkacik
                  <mailto:ttkacik@cisco.com>

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

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

description

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

Copyright (c) 2015 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-02; see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to draft-ietf-i2rs-yang-network-topo-02 with RFC number when published (i.e. RFC xxxx).";

```
revision 2015-12-08 {
  description
    "Initial revision.
    NOTE TO RFC EDITOR: Please replace the following reference
    to draft-ietf-i2rs-yang-network-topo-02 with
    RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-network-topo-02";
}

typedef node-id {
  type inet:uri;
  description
    "Identifier for a node.";
}

typedef network-id {
  type inet:uri;
  description
    "Identifier for a network.";
}
```



```

grouping network-ref {
  description
    "Contains the information necessary to reference a network,
    for example an underlay network.";
  leaf network-ref {
    type leafref {
      path "/nd:networks/nd:network/nd: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 "/nd:networks/nd:network[nd:network-id=current()/../"+
        "network-ref]/nd:node/nd: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 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 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 "/networks/network/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 "../..../supporting-network/network-ref";
                require-instance false;
            }
            description
                "References the underlay network that the
                underlay node is part of.";
        }
        leaf node-ref {
            type leafref {
                path "/networks/network/node/node-id";
            }
        }
    }
}

```



```
    prefix inet;
  }
  import ietf-network {
    prefix nd;
  }

  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:     Jeffrey Haas
                  <mailto:jhaas@pfr.org>

    Editor:       Alexander Clemm
                  <mailto:alex@cisco.com>

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

    Editor:       Robert Varga
                  <mailto:rovarga@cisco.com>

    Editor:       Tony Tkacik
                  <mailto:ttkacik@cisco.com>

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

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

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

    Copyright (c) 2015 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-02;
see the RFC itself for full legal notices.

NOTE TO RFC EDITOR: Please replace above reference to
draft-ietf-i2rs-yang-network-topo-02 with RFC
number when published (i.e. RFC xxxx).";

```
revision 2015-12-08 {
  description
    "Initial revision.
    NOTE TO RFC EDITOR: Please replace the following reference
    to draft-ietf-i2rs-yang-network-topo-02 with
    RFC number when published (i.e. RFC xxxx).";
  reference
    "draft-ietf-i2rs-yang-network-topo-02.";
}

typedef link-id {
  type inet:uri;
  description
    "An identifier for a link in a topology.
    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 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 on a node.
    The identifier SHOULD be chosen such that the same TP in a
    real network topology will always be identified through the
    same identifier, even if the model is instantiated in
    separate datastores. An implementation MAY choose to capture
    semantics in the identifier, for example to indicate the type
    of TP and/or the type of node and topology that the TP is a
    part of.";
}
```

```

grouping link-ref {
  description
    "References a link in a specific network.";
  leaf link-ref {
    type leafref {
      path "/nd:networks/nd:network[nd:network-id=current()/../"+
        "network-ref]/lnk:link/lnk: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 nd:network-ref;
}

grouping tp-ref {
  description
    "References a termination point in a specific node.";
  leaf tp-ref {
    type leafref {
      path "/nd:networks/nd:network[nd:network-id=current()/../"+
        "network-ref]/nd:node[nd:node-id=current()/../"+
        "node-ref]/lnk:termination-point/lnk: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 nd:node-ref;
}

augment "/nd:networks/nd:network" {
  description
    "Add links to the network model.";
  list link {
    key "link-id";
    description
      "A Network Link connects a by Local (Source) node and
      a Remote (Destination) Network Nodes via a set of the
      nodes' termination points.
      As it is possible to have several links between the same
      source and destination nodes, and as a link could
      potentially be re-homed between termination points, 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.

Layering dependencies on links in underlay topologies are not represented as the layering information of nodes and of termination points is sufficient."

```

container source {
  description
    "This container holds the logical source of a particular
    link.";
  leaf source-node {
    type leafref {
      path "../..../nd:node/nd:node-id";
    }
    mandatory true;
    description
      "Source node identifier, must be in same topology.";
  }
  leaf source-tp {
    type leafref {
      path "../..../nd:node[nd:node-id=current()]/../"+
        "source-node]/termination-point/tp-id";
    }
    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 "../..../nd:node/nd:node-id";
    }
    mandatory true;
    description
      "Destination node identifier, must be in the same
      network.";
  }
  leaf dest-tp {
    type leafref {
      path "../..../nd:node[nd:node-id=current()]/../"+
        "dest-node]/termination-point/tp-id";
    }
    description

```

```

        "Termination point within destination node that
        terminates the 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.";
}
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 "../..../nd:supporting-network/nd:network-ref";
            require-instance false;
        }
        description
            "This leaf identifies in which underlay topology
            supporting link is present.";
    }
    leaf link-ref {
        type leafref {
            path "/nd:networks/nd:network[nd: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 "/nd:networks/nd:network/nd: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
    "The leaf 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 "../..../nd:supporting-node/nd:network-ref";
    require-instance false;
  }
  description
    "This leaf identifies in which topology the
    supporting termination point is present.";
}
leaf node-ref {
  type leafref {
    path "../..../nd:supporting-node/nd:node-ref";
    require-instance false;
  }
  description
    "This leaf identifies in which node the supporting
    termination point is present.";
}
leaf tp-ref {
  type leafref {
    path "/nd:networks/nd:network[nd:network-id=current()/" +
      "../network-ref]/nd:node[nd: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>

5. Security Considerations

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

6. Contributors

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

- o Ken Gray, Cisco Systems
- o Xufeng Liu, Ericsson
- o Tom Nadeau, Brocade
- o Aleksandr Zhdankin, Cisco

7. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Alia Atlas, Vishna Pavan Beeram, Andy Bierman, Martin Bjorklund, Igor Bryskin, Benoit Claise, Susan Hares, Ladislav Lhotka, Carlos Pignataro, Juergen Schoenwaelder, and Xian Zhang.

8. References

8.1. Normative References

- [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.

- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6021] Schoenwaelder, J., "Common YANG Data Types", RFC 6021, October 2010.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

8.2. Informative References

- [I-D.draft-ietf-netconf-yang-push]
Clemm, A., Voit, E., and A. Gonzalez Prieto, "Subscribing to YANG datastore push updates", I-D draft-ietf-netconf-yang-push-00, October 2015.
- [I-D.draft-ietf-netmod-rfc6020bis]
Bjorklund, M., "The YANG 1.1 Data Modeling Language", I-D draft-ietf-netmod-rfc6020bis-08, October 2015.
- [topology-use-cases]
Medved, J., Previdi, S., Lopez, V., and S. Amante, "Topology API Use Cases", I-D draft-amante-i2rs-topology-use-cases-01, October 2013.

Authors' Addresses

Alexander Clemm
Cisco

EMail: alex@cisco.com

Jan Medved
Cisco

EMail: jmedved@cisco.com

Robert Varga
Cisco

EMail: rovarga@cisco.com

Tony Tkacik
Cisco

EMail: ttkacik@cisco.com

Nitin Bahadur
Bracket Computing

EMail: nitin_bahadur@yahoo.com

Hariharan Ananthakrishnan
Packet Design

EMail: hari@packetdesign.com