I2RS working group   S. Hares
Internet-Draft   Huawei
Intended status: Informational   A. Dass
Expires: September 13, 2017   Ericsson
                                                          March 12, 2017

NETCONF Changes to Support I2RS Protocol
draft-hares-netconf-i2rs-netconf-01.txt

Abstract

This document describes a NETCONF capabiilty to support the Interface to Routing system (I2RS) protocol requirements for I2RS protocol version 1. The I2RS protocol is a re-use higher layer protocol which defines extensions to other protocols (NETCONF and RESTCONf) and extensions to the Yang Data Modeling language.

The I2RS protocol supports ephemeral state datastores as control plane datastores. Initial versions of this document contain descriptions of the ephemeral datastore. Future versions may move this description to NETMOD datastore description documents.

Status of This Memo

This Internet-Draft will expire on September 13, 2017.

Copyright Notice

Table of Contents

1. Introduction

This a proposal for Yang additions to support the first version of the I2RS protocol.

The I2RS architecture [RFC7921] defines the I2RS interface "a programmatic interface for state transfer in and out of the Internet routing system". The I2RS protocol is a protocol designed to a higher level protocol comprised of a set of existing protocols which have been extended to work together to support a new interface to the routing system. The I2RS protocol is a "reuse" management protocol which creates new management protocols by reusing existing protocols and extending these protocols for new uses, and has been designed to be implemented in phases [RFC7921].

The first version of the I2RS protocol is comprised of extensions to existing features of NETCONF [RFC6241] and RESTCONF [I-D.ietf-netconf-restconf]. The data modeling language for the I2RS protocol will be Yang [RFC7950] with features and extensions proposed in this draft.

The structure of this document is:

Section 2 provides definitions for terms in this document.

Section 3 summarizes the I2RS requirements behind these changes.

Section 4 describes the NETCONF capability to support a control protocol datastore.

Section 5 the NETCONF capability to support ephemeral state. [I-D.ietf-i2rs-ephemeral-state] specifies the I2RS requirements for the ephemeral state.

Section 6 provides a Tiny Routing Rib Yang module used by the examples in this document.

2. Definitions Related to Ephemeral Configuration

This section reviews definitions from I2RS architecture [RFC7921] and NETCONF operational state definitions [I-D.ietf-netmod-revised-datastores] before using these to construct a definition of the ephemeral data store.

2.1. Requirements language

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

2.2. I2RS Definitions

The I2RS architecture [RFC7921] defines the following terms:

ephemeral data: is data which does not persist across a reboot (software or hardware) or a power on/off condition. Ephemeral data can be configured data or data recorded from operations of the router. Ephemeral configuration data also has the property that a system cannot roll back to a previous ephemeral configuration state. (See [RFC7921] for an architectural overview, [I-D.ietf-i2rs-ephemeral-state] for requirements, and [I-D.ietf-netmod-revised-datastores] for discussion of how the ephemeral datastore as a control plane datastore interacts with intended datstore and dynamic configuration protocols to form the applied datastore".

local configuration: is the data on a routing system which does persist across a reboot (software or hardware) and a power on/off condition. Local configuration has the ability to roll back to a pervious configuration state. Local configuration is defined as the intended datastore [I-D.ietf-netmod-revised-datastores] which is modified by dynamic configuration protocols (such as DHCP) and the I2RS ephemeral data store

dynamic configuration protocols datastore are configuration protocols such as DHCP that interact with the intended datastore (which does persist across a reboot (software or hardware) power on/off condition), and the I2RS ephemeral state control plane datastore.

operator-applied policy: is a policy that an operator sets that determines how the ephemeral datastore as a control plane data store interacts with applied datastore (as defined in [I-D.ietf-netmod-revised-datastores]). This operator policy consists of setting a priority for each of the following (per [I-D.ietf-i2rs-ephemeral-state]):

- intended configuration,
- any dynamic configuration protocols,
- any control plane datastores (one of which is ephemeral.)

3. Requirements control-plane datstore and ephemeral capabilities

3.1. I2RS protocol requirements

The requirements for the I2RS protocol are defined in the following documents:

- I2RS Problem Statement [RFC7920],
- I2RS Architecture [RFC7921],
- I2RS Traceability [RFC7922],
- Publication and Subscription [RFC7923],
- I2RS Ephemeral State Requrements, , [I-D.ietf-i2rs-ephemeral-state]
- I2RS Protocol Security Requirements, [I-D.ietf-i2rs-protocol-security-requirements]

The Interface to the routing System (I2RS) creates a new capability for the routing systems, and with greater capaiblities come a greater need for security. The requirements for a secure environment for I2RS are described in [I-D.ietf-i2rs-security-environment-reqs].

3.2. Overview of NETCONF support for I2RS protocol requirements

This oveview reviews the following:

- Dependencies on Existing features
- Additions to use NETCONF [RFC6241] to support control plane datastores changes get to get data, write data,(via target [merge, replace, create, delete, copy, delete-all]), close-session, kill-session, rollback-on-error (all-or-nothing), validate (validation + roll-back-on-error (all-or-nothing),
- Additions for I2RS ephemeral
- NETCONF [RFC6241] changes to obtain control-plane datastore.

4. NETCONF capability for control-plane datastore

capability-name: control-plane

4.1. Overview

This capability defines the NETCONF protocol extensions for use with control plane protocols.

A control plane datastore is not part of the configuration datastore per [I-D.ietf-netmod-revised-datastores]. The control plane datastore many contain configuration and operational state. A router implementation may merge the configuration from a control plane datastore with configuration data from the configuration datastore. A query of the applied datastore will provide a list of the installed configuration from all datastores with meta data. The current architectural provides an origin identityref with the following mappimg to datastores for the

- static - configuration data store.
- dynamic - dynamic configuration or dynamic control plane datastores.
- system - created by system,
- data-model - created by "default" in use.

Clearly, the dynamic origin title is not enough to uniquely identify a control plane datastore entry in the applied datastore. Additional definitions will need to be added to the architectural model, but this will be specified in another document.

The control plane datastores do not restrict multiple access via the locking mechanisms (<lock> and <unlock>), but use a priority scheme to handle multiple clients attempting to write the same data. The default validation within a control plane datastore’s config objects (e.g. config=TRUE) is the configuration datastore validation, but if Yang data modules specify different validation for the datastore or specific nodes then the control plane datastores will use this validation.

Some data modules may be used for both a control plane datastore and the configuration datastore. If additional validation is used for these modules, it is recommend that these modules use the "rpc" function for the additional validation rather than the <write-data> functions.

4.2. Dependencies

The following are the dependencies for the :control-plane capability

- Yang features:
  - [I-D.ietf-netmod-revised-datastores] functionality including the ietf-yang-architecture" data module.
  - [I-D.hares-netmod-i2rs-yang] Yang additions related to datastores definitions related to control plane datastores (datastoredef, datastore, dstype, precedence, protosup, validation), and ephemeral state.
- The following NETCONF features:
  - NETCONF [RFC6241] with its updates [RFC7803],
  - Network Access Control Model [RFC6536] with update by [I-D.ietf-netconf-rfc6536bis]
  - Running NETCONF over TLS with mutually X.509 authentication [RFC7589]
  - Keystore Model [I-D.ietf-netconf-keystore],
  - Subscribing to Yang Datastore updates [I-D.ietf-netconf-yang-push],
  - NETCONF support for Event Notifications [I-D.ietf-netconf-netconf-event-notifications],
  - Subscribing to NETCONF Events (updated) [I-D.ietf-netconf-rfc5277bis]
  - Yang Patch Media type [I-D.ietf-netconf-yang-patch],
  - NETCONF/RESTCONF Zero Touch provisioning [I-D.ietf-netconf-zerotouch],
  - TLS Client and Server Models [I-D.ietf-netconf-tls-client-server]
  - Call Home [I-D.ietf-netconf-call-home],
  - Module library [RFC7895],

  - NETCONF/RESTCONF Zero Touch provisioning [I-D.ietf-netconf-zerotouch],

4.3. Capability identifier

The controlplane-datastore capability is identified by the following capability string: (:control-plane (uri-tbd)) where the uri-tbd is to be assigned by IANA.

4.4. New Operations

The following are additional protocol operations NETCONF [RFC6241] to support the following queries based on a datastore source/target datastore being specified:

- "get-data"

- "write-data"

- "validate-data"

The <target-datastore> must be registered with IANA.

4.4.1. <get-data>

The get-data command has obtains configuration and operational data. The parameters the following:

source name of the datastore being queried. The valid names are "applied", "opstate", or a datstore name registered with IANA.

filter this identifies the portions of the device configuration datastore is to receive. If this parameter is not present, the entire datastore is returned. The filter MAY support subtypes "subtree", "uri", and "xpath" capabilities described in [RFC6241]. Filters may also include the elements for state (E.g. config true, config false, ephemeral true; ephemeral false;).

Positive Response If the device was able to satify the request, an <rpc-reply> is sent. The <data> section contains the appropriate subset.

Negative Response If the device was unable to satify the request, an <rpc-error> is included in the <rpc-reply>

Example - retrieve route1 in route list.
wfrom control plane datastore (cp-alpha)
and gets both configuration and ephemeral data.

```
 <rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data/
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                  <route-list>
                     <route-index>1</route-index>
                  </route-list<
      </filter>
    </get-data>
 </rpc>

  <rpc-rply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data<
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                  <route-list>
                     <route-index>1</route-index>
                         <prefix-match>192.2/8 /16<prefix-match>
                         <nexthop>129.1.5.1</nexthop>
                         <if-outgoing>Eth0</if-outgoing>
                         <installed>true</installed>
                  </route-list<
      </filter>
    </get-data>
 </rpc>
```

Figure 1

Example 2 - retrieve users subtree from the ephemeral database which has example control plane datastore (cp-alpha) and gets only config=true data;

```
 <rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data/
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                <route-list>
                    <route-index>1</route-index>
                 </route-list<
                type="state" select "config";
      </filter>
    </get-data>
 </rpc>

  <rpc-rply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data<
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                 <route-list>
                    <route-index>1</route-index>
                        <prefix-match>192.2/8 /16<prefix-match>
                        <nexthop>129.1.5.1</nexthop>
                        <if-outgoing>Eth0</if-outgoing>
                 </route-list<
      </filter>
    </get-data>
 </rpc>
```

Figure 2 - get config data

4.4.2. <write data gt;

The write data operationals has the attributes of operation parameters, and parameters of target datastore, default-operations, test-option, error-option, a priority, secondary-id, config, and opstate. The attributes of the operation for individual nodes wutg "config-true" are: create, delete, merge, remove, and replace. The attributes for all-datastore operations are create-datastore, copy-datastore, delete-datastore. The operations handle multiple-client writes by using priority values rather than a locking mechanism. If two or more clients interact over changing the data node, the priority values arbitrate between the the clients where the greatest priority (1=lowest) wins. The following operations can enact changes in opstate data nodes: create, delete, remove, reset.

The default-operations parameter flags are: merge, replace, or none. The test-option parameters flags are: test-then-set, set, and test-only. The error-option oarameter flags are: stop-on-error, continue-on-error, and rollback-on-error. The priority parameter is a integer giving the priority (range 1..5000).

4.4.2.1. Limitations based on other capability flags

The test-option parameters MAY only be set if the device advertises the :validate:1.1 capability. If test-option is set without the :validate:1.1 capability, an error is returned "no support for test-option".

The error-option subparameter "rollback-on-error" is enabled only if the :rollback-on-error capability is set and the data is under the config parameter.

A URL is accepted within the <source> or <target> parameters if the :url capability is set. An XPATH is accepted within the <source> or <target> parameters if the :xpath capability is set.

4.4.2.2. defaults

The following are the defaults:

- The target datastore does not exists, it will be created if local support exists. Otherwise, the reply will indicate "non-supported datastore".
- If no sub-operations is specified the sub-operation, the default is merge.
- If no priority parameter is specified, the priority will be set to 1 (lowest external priority).
- If error-option is not set, then the default is "stop-on-error". (Note: for I2RS WG input. Is "stop-on-error" the same as "none"?)

- If no test-option parameter is set, the write data operates as a ’set" without validation first.

- if no secondary-identifier is given, the secondary identifier stored is "" indicating a null string.

The NETCONF priority for the "write data" function is simply the Netconf priority range. If implementations have an internal priority, the precedence between the local configuration and the NETCONF supplied configuration is set by a operator applied knob. For example, an implementation could indicate that the local configuration priority can range from 0-10 and the NETCONF priority is 10 plus the value of the priority parameter.

4.4.2.3. target

The target is a datastore whose name is registered with IANA.

Example Write data

dsname = i2rs-agent

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <write data>
    <target><i2rs-agent></target>
        <operation>merge</operation>
    <priority>50<</priority>
        <error-option>all-or-nothing</error-option>
        <config>
             <top xmlns:t=
          "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                  <route-list>
                     <route-index>1</route-index>
                         <prefix-match>192.2/8 /16<prefix-match>
                         <nexthop>129.1.5.1</nexthop>
                         <if-outgoing>Eth0</if-outgoing>
                  </route-list<
              </top>
        </config>
   </write data>
```

4.4.2.4. write data operations attributes

The description of each operation is below:

<create> (config, opstate) : the creation of the data node in the datatstore if and only if the data does not already exist in the target datastore. If the data already exists, then an <rpc-error> is return wtih an error tag of "data-exists". Failure information or Success informatnoi is also pass to the notification functions (events and traceability).

<create-datastore > (config, opstate) : the creation of the datastore based on datastructures in the config and opstate parameters. The datastore ownership is set to client creating the datastore plus the priority.

<delete> (config) : the deletion of the data node if the data node exists in the configuration and either the same client deletes the node or a client with a high priority deletes the node. If configuration data does not exist in the datastore, then the <rpc-error> element is returned with a <error-tag> with value of "data-missing". The error information is passed to the notification functions to be sent as event or (optionally) placed in a tracing file.

<delete-datastore> Delete all configuration and operational data configured into the datastore, and the delete the datastore. The client requesting a delete-store must either be the owner of the datastore or have a higher priority than the client that owns the datastore. If a higher priority client takes ownership, the lower priority client is notified. If the devices is able to satisfy request, the positive response is <rcp-reply> that includes <ok> element. If the device is unable to complete request, the <rcp-reply> that includes <rpc-error> element. The operations results are forwarded to event and traceabilty functions.

<copy-datastore> If the datastore does not exist, it creates the datastore and copies the configuration values and opstate values into the datastore. The ownership information (client identity and priority) is saved as part of the datastore. If the datastore does exist and the client with ownership of the datastore changes it, then the client can replace all the datastore nodes. If a different client with lower priority than the client having ownership wants to change the datastore, the request is rejected. If a client with higher priority than the client having ownership, then the the owership changed to the new client, all the data in the datastore is deleted, all new data uploaded (config and opstate nodes). If a server is able to satisfy request, the

positive response is <rcp-reply> that includes <ok> element. If the server is unable to complete request, the <rcp-reply> that includes <rpc-error> element. The operational results are forwarded to event and traceabilty functions. If a copy-datastore action is in progress, and a client with a higher priority asks to copy-datastore, the original

<merge> (config) : parameter specifies merge. If the <priority< specifies The current data is modified by the new data in a merge of the data based on priorities. If the same client merges the data, priority is ignored. If a different client merges the data, the priority must be created than the current client’s priority. If any data is replaced, this event is passed to the notification and traceability functions to pass to the setting client and the client that set the original value.

<remove> (config, opstate) : the remove of the data node if the data nodes specified in the <config> or the <opstaste> node exist. If data nodes do not exist, the "remove" operation is silently ignored and error results are forwarded to traceabilty functions.

<replace> (config) : replaces data in target if the same client replaces the top-level node, priority is ignored. If a different client replaces the note, the priority must be higher than the top level node’s priority. If any data is replaced, this event is passed to the notification function (events and traceability), and a notification is sent to the previous client setting this data that the data has been reset. If the request to replace is reject due to the current top-level node having a higher priority, then an <rpc-error> returns with an error tag of "insufficient-priority". If the node is replace by a different client, the original client is notified of the change.

<reset> (opstate) : resets opstate nodes with counters to initial settings.

4.4.2.5. <priority parameters>

The priority parameter sets a integer value for the priority as shown in figure x.

4.4.2.6. <test-op parameter>

The <test-option> parameter performs basic function it does in the <edit-config> basic function. Just in [RFC6241], the <test-option> parameter MAY be specified only if the device advertises the ":validate:1.1" capability. The only difference is that the validation specified by the data model may augment the validation test and the valdiation will also include the ability of the client to set this element. If a validation error occurs, the test-then-set will not perform the write-data function.

4.4.2.7. <error-option> parameter

<error-option> has the following attributes

stop-on-error : Abort the write-data on the first error. This is the default.

msg-rollback-on-error : if an error condition occurs such that error serverity <rpc-error> is generated, the server will stop processing the write-data operation and restore the specific configuratoin to its complete state at the start of the "write-data" operation. This option only processes roll-back on single messages which includes

- if multiple operations occur in single message, error in one operation (E.g. read data) must not impact other operations (write-data);

- multiple operations in multiple message should be supported, but roll-back should only include a single message.

This option requires the server to support the :rollback-on-error capability.

4.4.2.8. secondary-id

This operation associates a secondary identifier with a set of write-data operations. The secondary identifier is an opaque string.

4.5. Modification to protocol operations

4.5.1. Unsupported protocol operations

The following protocol operations are not supported in the control plane datastore:

- <get-config>,
- <edit-config>,
- <copy-config>,
- <delete-config>,

- <lock>,
- <unlock>

4.5.2. Modified protocol operations

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

The <close-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

The <kill-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

4.6. Interactions with Capabilities

4.6.1. Unsupported Capabiltiies

The following capabilities are not supported:

writeable-running capability,

candidate configuration capability,

confirmed commit capability,

distinct startup capbility,

4.6.2. Modified Capabilities

4.6.2.1. rollback-on-error

The rollback-on-error allows the error handling to be roll-back-on-error (all-or-nothing in I2RS terms) for the control plane datastore. The control plane datastore name is a valid target if the rollback-on-error capability is combined with the control plane datastore capability.

4.6.2.2. validate

The validation capability engated with the control plane capability operates to validate the config portion of the control plane datastore. Therefore, the <target> is allowed to have a datstore name which is registered with IANA.

The validation of the configuration portion may contain the "validation" yang command which provides alternative validation mechanisms for specific data objects.

4.6.2.3. URL capability and XPATH capability

The URL capabilities specify a <url> in the <source> and <target> operate as normal, but are allowed to specify a module within a control plane datastore.

5. NETCONF Ephemeral capability

capability-name: control-plane

5.1. Overview

The ephemeral capability is the ability to support control plane datastores which are entirely ephemeral or have ephemeral state modules, or ephemeral statements within objects in a modules. These objects can be configuration state (config=TRUE) or operational state (config=FALSE).

Ephemeral state in datastores, ephemeral modules or ephemeral objects within a module have one key characteristics: the data does not persist across reboots. The ephemeral configuration state must be restored by a client, and the operational state will need to be regenerated.

The entire requirements for ephemeral state for the I2RS control plane protocol are listed in [I-D.ietf-i2rs-ephemeral-state]. Many of these require fulfilled by the NETCONF control-plane capabilty(Ephemeral-REQ-07, Ephemeral-REQ-11, Ephemeral-REQ-12, Ephemeral-REQ-13, Ephemeral-REQ-14, Ephemeral-REQ-16).

The key features include:

- references between (to/from) ephemeral state and non-ephemeral state for constraints purposes (see Ephemeral-REQ-02, Ephemeral-REQ-03, and Ephemeral-REQ-04 in [I-D.ietf-i2rs-ephemeral-state]).
- operations to set and modify the constraints on the amount of resources the I2RS Agent (aka NETCONF server) can consume (Ephemeral-REQ-05)
- Yang modules must identify Yang objects (modules, submodules or objects within yang modules which are ephemeral and augment other nodes) and allow an "ephemeral=TRUE" feature.

5.2. Dependencies

Ephemeral state is not supported in the configuration datstore. The ephemeral state capability depends on having the control-plane datastore capability enabled (with appropriate NETCONF capabilities described above), and an IANA registered datastore name.

Yang must support the ability to to denote that a datastore, module, submodule or object within a module can be denoted as ephemeral. This capbility depends on the yang additions described in [I-D.hares-netmod-i2rs-yang] for control plane datastores, ephemeral key word, and validation key word.

Ephemeral state operation depends on notification of events and traceability of errors. I2RS ephemeral state requires that

5.3. New Operations

Note: One operation that is suggested for ephemeral state is to set resource limits. It does not seem to be an ephemeral state issue, but a control plane issue. This feature is placed here until future discussion for I2RS WG.

5.3.1. resource-limits

resource-limits

definition - TBD

The [I-D.ietf-i2rs-ephemeral-state] suggests setting these limits, but it does not seem to be an ephemeral function.

5.4. Modifications to Protocol Operations

5.4.1. Unsupported Operations

The ephemeral state only works as an augment to the control-plane datastore. Therefore, the following protocol operations, which are not supported in the control-plane datastore capability, are also not supported in the ephemeral capability:

- <get-config>,
- <edit-config>,
- <copy-config>,
- <delete-config>,

- <lock>,
- <unlock>

5.4.2. Modified Operations

The ephemeral state only works as an augment to the control-plane datastore with specific ephemeral validations. Therefore, the <close-session> and <kill-session> are modified as described in the sections below.

5.4.2.1. <close-session> Modifications

The <close-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

5.4.2.2. <close-session> Modifications

The <kill-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

5.4.2.3. <validate> Modifications

The ephemeral state may require validation to determine if the constraints obey ephemeral-state rules. If the :validate capability is used, the following parameter requires ephemeral-state contraints (Ephemeral-REQ-02, Ephemeral-REQ-03, and Ephemeral-REQ-04). If the ephemeral-constraint parameter is engaged for a module or object that is not ephemeral, the parameter is silently ignored. Error information is forwarded to the event notification processes and the traceability functions.

Additional Parameter

ephemeral-constraint

6. Yang model Simple Ephemeral Data model

```
datastoredef cp-alpha {
   dstype control-plane;
   description {
          "example control plane datastore ";
     }
   module-list tiny-rt-instance;
   precedence applied {
     precedenceval 5;
```

```
      }
      precedence opstate {
         precedenceval 5;
      }
   }

   datastoredef cp-beta {
      dstype control-plane i2rs-v0;
      description {
             "example control plane datastore ";
        }
      module-list tiny-rib;
      ephemeral true;
      precedence applied {
        precedenceval 50;
      }
      precedence opstate {
         precedenceval 50;
      }
   }
   module cp-example-1 {
    namespace "http://exaple.com/schema/cp-examples/1.1/tiny-rib";
    prefix trib;
    import ietf-inet-types {
       prefix inet;
    }
    import ietf-yang-types {
       prefix yang;
     }

    grouping trib-rt {
       description "tiny rib route";
       leaf route-index  {
              type uint64;
              mandatory true;
              description "route index";
            }
           leaf v4-prefix-match {
                  type inet:ipv4-prefix;
                  mandatory true;
           }
           leaf v4-nexthop {
               type inet:ipv4
                   mandatory true;
              }
           leaf if-outgoing {
               type if:interface-ref;
                   mandatory true;
```

```
                   description {
                    "Name of outgoing interface";
                   }
           leaf installed {
                   type boolean;
           config false;
                   description "rt install status ";
              }
           }
      container tiny-rt-instance {
        description
               "Tiny routing instance for
                    example purposes";
            leaf name {
               type string;
                   description
                    "The name of routing instance
             which must be unique. ";
        }
            list route-list {
              key "route-index";
              description
                "a list of routes of rib"
              uses trib-rt;
            }
      }

      rpc trib-add {
       description "add route to tiny rib";
           input {
             leaf datastore {
            type string; //iana registered
            mandatory true;
            description
                      "iana datastore name";
             }
             container trib-routes {
                description
                      "Tiny rib routes to be added
                       to tiny rib";
                    list route-list {
                      key "route-index";
                      users trib-rt;
                    }
             }
        }
            output {
              container trib-add-status {
```

```
                leaf success {
                     type boolean;
                     description "add succeded";
                    }
                    leaf failure-reason {
                     type string;
                     description "reason for failure ";
                    }
              }
       }
    }
```

7. IANA Considerations

TBD

8. 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.

9. Acknowledgements

TBD

10. References

10.1. 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>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC5339] Le Roux, JL., Ed. and D. Papadimitriou, Ed., "Evaluation of Existing GMPLS Protocols against Multi-Layer and Multi-Region Networks (MLN/MRN)", RFC 5339, DOI 10.17487/RFC5339, September 2008, <http://www.rfc-editor.org/info/rfc5339>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[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>.

[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>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

[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>.

[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>.

[RFC7803] Leiba, B., "Changing the Registration Policy for the NETCONF Capability URNs Registry", BCP 203, RFC 7803, DOI 10.17487/RFC7803, February 2016, <http://www.rfc-editor.org/info/rfc7803>.

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

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

[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, <http://www.rfc-editor.org/info/rfc7921>.

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

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

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <http://www.rfc-editor.org/info/rfc7952>.

[RFC7958] Abley, J., Schlyter, J., Bailey, G., and P. Hoffman, "DNSSEC Trust Anchor Publication for the Root Zone", RFC 7958, DOI 10.17487/RFC7958, August 2016, <http://www.rfc-editor.org/info/rfc7958>.

10.2. Informative References

[I-D.hares-netmod-i2rs-yang]
Hares, S. and a. amit.dass@ericsson.com, "Yang for I2RS Protocol", draft-hares-netmod-i2rs-yang-04 (work in progress), March 2017.

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-23 (work in progress), November 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-17 (work in progress), September 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-07 (work in progress), January 2017.

[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-10 (work in progress), December 2016.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-03 (work in progress), March 2017.

[I-D.ietf-i2rs-yang-l3-topology]
Clemm, A., Medved, J., Varga, R., Liu, X., Ananthakrishnan, H., and N. Bahadur, "A YANG Data Model for Layer 3 Topologies", draft-ietf-i2rs-yang-l3-topology-08 (work in progress), January 2017.

[I-D.ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), December 2015.

[I-D.ietf-netconf-keystore]
Watsen, K. and G. Wu, "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), October 2016.

[I-D.ietf-netconf-netconf-event-notifications]
Prieto, A., Clemm, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF Support for Event Notifications", draft-ietf-netconf-netconf-event-notifications-01 (work in progress), October 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[I-D.ietf-netconf-rfc5277bis]
Clemm, A., Prieto, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "Subscribing to Event Notifications", draft-ietf-netconf-rfc5277bis-01 (work in progress), October 2016.

[I-D.ietf-netconf-rfc6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-00 (work in progress), January 2017.

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-01 (work in progress), November 2016.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-14 (work in progress), November 2016.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-05 (work in progress), March 2017.

[I-D.ietf-netconf-zerotouch]
Watsen, K. and M. Abrahamsson, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-12 (work in progress), January 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", draft-ietf-netmod-revised-datastores-00 (work in progress), December 2016.

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-04 (work in progress), March 2017.

[I-D.ietf-netmod-syslog-model]
Wildes, C. and K. Koushik, "A YANG Data Model for Syslog Configuration", draft-ietf-netmod-syslog-model-12 (work in progress), February 2017.

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: Informational
Expires: September 30, 2017

S. Hares
Huawei
A. Dass
Ericsson

March 29, 2017

RESTCONF Changes to Support I2RS Protocol
draft-hares-netconf-i2rs-restconf-02.txt

Abstract

This document describes two RESTCONF optional capabilities (i2rs-control plane capability, ephemeral state capabilities) that are needed to support the I2RS protocol needs.

The purpose of this draft is to kick-start the discussions with I2RS Working Group and NETCONF WG on these two capabilities.

Status of This Memo

This Internet-Draft will expire on September 30, 2017.

Copyright Notice

Table of Contents

1. Introduction

This a proposal for the following two RESTCONF capabilities to augment RESTCONF [RFC8040] to support the first version of the I2RS protocol: Control plane datstore capability and ephemeral state capability. The yang that supports this proposal is described in [I-D.hares-netmod-i2rs-yang]. This work is based on the datastore definitions in [I-D.ietf-netmod-revised-datastores].

This draft parallels a similar proposal for NETCONF [RFC6241] is described in [I-D.hares-netconf-i2rs-protocol]. One difference between the proposed capabilities for i2rs control-plane capability additions to NETCONF and the proposed capabilities for i2rs control-plane for RESTCONF is write-collection. RESTCONF has edit-collision capability already which only needs a usage description.

1.1. Background on I2RS

The I2RS architecture [RFC7921] defines the I2RS interface "a programmatic interface for state transfer in and out of the Internet routing system". The I2RS protocol is a protocol designed to a higher level protocol comprised of a set of existing protocols which have been extended to work together to support a new interface to the routing system. The I2RS protocol is a "reuse" management protocol which creates new management protocols by reusing existing protocols and extending these protocols for new uses, and has been designed to be implemented in phases [RFC7921].

1.2. Structure of draft

The structure of this document is:

Section 2 provides definitions and background on I2RS work. (If you are familiar with the I2RS architecture and requirements, you can skip this section.)

Section 3 describes the RESTCONF control plane datastore capability.

Section 4 describes the RESTCONF ephemeral state capabiilty. .

2. Definitions and Background on I2RS

This section reviews definitions from I2RS architecture, and provides background on I2RS work for the reader.

2.1. IETF Requirements language

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

2.2. I2RS Definitions

The I2RS architecture [RFC7921] defines the following terms:

ephemeral data: is data which does not persist across a reboot (software or hardware) or a power on/off condition. Ephemeral data can be configured data or data recorded from operations of the router. Ephemeral configuration data also has the property that a system cannot roll back to a previous ephemeral configuration state. (See [RFC7921] for an architectural overview, [I-D.ietf-i2rs-ephemeral-state] for requirements, and [I-D.ietf-netmod-revised-datastores] for discussion of how the ephemeral datastore as a control plane datastore interacts with intended datstore and dynamic configuration protocols to form the applied datastore".

local configuration: is the data on a routing system which does persist across a reboot (software or hardware) and a power on/off condition. Local configuration has the ability to roll back to a pervious configuration state. Local configuration is defined as the intended datastore [I-D.ietf-netmod-revised-datastores] which is modified by dynamic configuration protocols (such as DHCP) and the I2RS ephemeral data store

dynamic configuration protocols datastore are configuration protocols such as DHCP that interact with the intended datastore (which does persist across a reboot (software or hardware) power on/off condition), and the I2RS ephemeral state control plane datastore.

control plane protocols datastore is a datastore which is loaded by control plane protocols (e.g. I2RS protocol) rather than system configuration protocols. (see [I-D.ietf-netmod-revised-datastores]).

operator-applied policy: is a policy that an operator sets that determines how the ephemeral datastore as a control plane data store interacts with applied datastore (as defined in [I-D.ietf-netmod-revised-datastores]). This operator policy consists of policy knobs that the operator sets to determine how the I2RS agent control plane ephemeral state datastore will interact with the intended configuration datastor and the dynamic configuration protocol datastore. Three policy knobs could be used to implement this policy:

- policy knob 1: I2RS Ephemeral control-plane datastore takes takes precedence over the intended datastore in the routing protocols.

- policy knob 2: Updated intended configuration datastore takes precedence over the I2RS ephemeral control-plane data store in the routing protocols

- policy knob 3: Ephemeral control plane datastore takes precedence over any other dynamic configuration protocols datastore.

2.3. I2RS protocol requirements

The requirements for the I2RS protocol are defined in the following documents:

- I2RS Problem Statement [RFC7920],
- I2RS Architecture [RFC7921],
- I2RS Traceability [RFC7922],
- Publication and Subscription [RFC7923],
- I2RS Ephemeral State Requrements, , [I-D.ietf-i2rs-ephemeral-state]
- I2RS Protocol Security Requirements, [I-D.ietf-i2rs-protocol-security-requirements]

The Interface to the routing System (I2RS) creates a new capability for the routing systems, and with greater capaiblities come a greater need for security. The requirements for a secure environment for I2RS is described in [I-D.ietf-i2rs-security-environment-reqs].

3. RESTCONF control plane datastore capability

capability-name: i2rs-control-plane

3.1. Overview

The i2rs-control-plane datastore capability enables the RESTCONF to support the following dynamic control plane datastore.

- API resource that is {+restconf}/datastore/<datastore-name>/data/ and operational state specific to the control plane datastore ({+restconf/cp-data/opstate}).
- It also includes the ability to have the applied datastore and the opstate datatstore (per [I-D.ietf-netmod-revised-datastores]) with the ability to return meta-data with the following information:
  - Entity-Tag encoding of <client-id><priority> or any portion of the filter.
  - "with defaults"
  - "with validation" - Yang specified validation (Unclear if this is the best way for validation.)

Ability to provide read access for the configuration datstore

Ability to provide read access for other dynamic datastores

3.2. Dependencies

This protocol strawman utilizes the following existing proposed features for NETCONF and RESTCONF

- RESTCONF [RFC8040].
- Module library [RFC7895],
- RESTCONF Patch Media Type [RFC8072],
- NETCONF Support for event notifications [I-D.ietf-netconf-netconf-event-notifications],
- Publication/Subscription via Push [I-D.ietf-netconf-yang-push],
- NETCONF and HTTP Transport for Event Notivications [I-D.ietf-netconf-restconf-notif],
- Publication/Subscription via Push [I-D.ietf-netconf-yang-push],
- syslog yang module (both [RFC5424] and [I-D.ietf-netmod-syslog-model]

3.3. New Operations

none

3.4. Modified Operations

All RESTCONF methods (OPTIONS, HEAD, GET, POST, PT, PATCH, DELETE) need to work in the control plane datastores. config=TRUE data, and where appropriate config=FALSE data.

4. RESTCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-state

4.1. Overview

This capability defines the RESTCONF protocol extensions for control plane protocols that support control plane data stores with ephemeral data.

Ephemeral state is not unique to I2RS work.

The ephemeral capability is the ability to support a dynamic datastores which are entirely ephemeral or have ephemeral state modules, or ephemeral statements within objects in a modules. These objects can be configuration state (config=TRUE) or operational state (config=FALSE).

Ephemeral state in datastores, ephemeral modules or ephemeral objects within a module have one key characteristics: the data does not persist across reboots. The ephemeral configuration state must be restored by a client, and the operational state will need to be regenerated.

The entire requirements for ephemeral state for the I2RS control plane protocol are listed in [I-D.ietf-i2rs-ephemeral-state]. Compared to RESTCONF functionality there are 4 groups of additional changes:

Constraints The ability to enforce the constraints for get (aka read) references (to/from) the {+restconf/data} datastore, and {+restconf/cp-data} control plane datastore. ((see Ephemeral-REQ-02, Ephemeral-REQ-03, and Ephemeral-REQ-04 in [I-D.ietf-i2rs-ephemeral-state]). The "validation" yang statement in [I-D.hares-netmod-i2rs-yang] could encode specific validation for the ephemeral case per datatstore or per object. [Editor’s note: Aid is needed to determine how validation occurs.]

Ephemeral in Data Modules Yang modules must identify Yang objects (modules, submodules or objects within yang modules which are ephemeral and augment other nodes) and allow an "ephemeral=TRUE" feature.

Roll-back an ephemeral node cannot roll-back to its previous value,

4.2. Dependencies

The ephemeral capabilities have the following dependencies:

- Yang modules must support the following:
  - identifying datastores, modules, and objects as ephemeral. (ephemeral=True)
  - Ability to have control plane datastores which are ephemeral.
- The following features must be supported by RESTCONF

- Module library [RFC7895],
- RESTCONF Protocol [RFC8040],
- RESTCONF Patch Media Type [RFC8072],
- NETCONF Support for event notifications [I-D.ietf-netconf-netconf-event-notifications],
- Publication/Subscription via Push [I-D.ietf-netconf-yang-push],
- NETCONF and HTTP Transport for Event Notivications [I-D.ietf-netconf-restconf-notif],
- Subsribing to Yang datastore push updates [I-D.ietf-netconf-yang-push],

4.3. Capability identifier

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

4.4. New Operations

none

4.5. Modification to data resources

RESTCONF must be able to support the ephemeral data in an an control-plane dynamic datastore. This is any API resource that is {+restconf}/datastore/<datastore-name>/data/ and operational state specific to the control plane datastore ({+restconf/cp-data/opstate}).

RESTCONF library functions must be able to store an indication that a data module has ephemeral state as meta-data.

4.6. Modification to existing operations

RESTCONF operations of GET, POST, PUT, PATCH, and DELETE must be able to filter on meta-data with "ephemeral" flag. (Should this be only read).

The operations must support the following things about ephemeral.

1. The ephemeral does not persist over a reboot,
2. an ephemeral node cannot roll-back to its previous value,

5. IANA Considerations

TBD -

6. 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.

7. Acknowledgements

TBD

8. References

8.1. Normative References:

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

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC5339] Le Roux, JL., Ed. and D. Papadimitriou, Ed., "Evaluation of Existing GMPLS Protocols against Multi-Layer and Multi-Region Networks (MLN/MRN)", RFC 5339, DOI 10.17487/RFC5339, September 2008, <http://www.rfc-editor.org/info/rfc5339>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[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>.

[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>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

[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>.

[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>.

[RFC7803] Leiba, B., "Changing the Registration Policy for the NETCONF Capability URNs Registry", BCP 203, RFC 7803, DOI 10.17487/RFC7803, February 2016, <http://www.rfc-editor.org/info/rfc7803>.

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

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

[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, <http://www.rfc-editor.org/info/rfc7921>.

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

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

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <http://www.rfc-editor.org/info/rfc7952>.

[RFC7958] Abley, J., Schlyter, J., Bailey, G., and P. Hoffman, "DNSSEC Trust Anchor Publication for the Root Zone", RFC 7958, DOI 10.17487/RFC7958, August 2016, <http://www.rfc-editor.org/info/rfc7958>.

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

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <http://www.rfc-editor.org/info/rfc8072>.

8.2. Informative References

[I-D.hares-netconf-i2rs-protocol]
Hares, S. and a. amit.dass@ericsson.com, "NETCONF Changes to Support I2RS Protocol", draft-hares-netconf-i2rs-protocol-00 (work in progress), November 2016.

[I-D.hares-netmod-i2rs-yang]
Hares, S. and a. amit.dass@ericsson.com, "Yang for I2RS Protocol", draft-hares-netmod-i2rs-yang-04 (work in progress), March 2017.

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-23 (work in progress), November 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-17 (work in progress), September 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-07 (work in progress), January 2017.

[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-10 (work in progress), December 2016.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-05 (work in progress), March 2017.

[I-D.ietf-i2rs-yang-l3-topology]
Clemm, A., Medved, J., Varga, R., Liu, X., Ananthakrishnan, H., and N. Bahadur, "A YANG Data Model for Layer 3 Topologies", draft-ietf-i2rs-yang-l3-topology-08 (work in progress), January 2017.

[I-D.ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), December 2015.

[I-D.ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-01 (work in progress), March 2017.

[I-D.ietf-netconf-netconf-event-notifications]
Prieto, A., Clemm, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF Support for Event Notifications", draft-ietf-netconf-netconf-event-notifications-01 (work in progress), October 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[I-D.ietf-netconf-restconf-notif]
Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Clemm, A., and A. Bierman, "Restconf and HTTP Transport for Event Notifications", draft-ietf-netconf-restconf-notif-02 (work in progress), March 2017.

[I-D.ietf-netconf-rfc5277bis]
Clemm, A., Prieto, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "Subscribing to Event Notifications", draft-ietf-netconf-rfc5277bis-01 (work in progress), October 2016.

[I-D.ietf-netconf-tls-client-server]
Watsen, K. and G. Wu, "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-02 (work in progress), March 2017.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-14 (work in progress), November 2016.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-05 (work in progress), March 2017.

[I-D.ietf-netconf-zerotouch]
Watsen, K. and M. Abrahamsson, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-13 (work in progress), March 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-01 (work in progress), March 2017.

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-04 (work in progress), March 2017.

[I-D.ietf-netmod-syslog-model]
Wildes, C. and K. Koushik, "A YANG Data Model for Syslog Configuration", draft-ietf-netmod-syslog-model-14 (work in progress), March 2017.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com

I2RS working group   S. Hares
Internet-Draft   Huawei
Intended status: Informational   A. Dass
Expires: September 12, 2017   Ericsson
                                                          March 11, 2017

Yang for I2RS Protocol
draft-hares-netmod-i2rs-yang-04.txt

Abstract

This document requests yang language additions for the data models that exist as part of the I2RS control plane datastore. One of these additions is the ability to mark a portion of the model as having ephemeral state.

Status of This Memo

This Internet-Draft will expire on September 12, 2017.

Copyright Notice

Table of Contents

1. Introduction

This a proposal for additions to yang 1.1 [RFC7950] to support the I2RS protocol.

The I2RS architecture [RFC7921] defines the I2RS interface "a programmatic interface for state transfer in and out of the Internet routing system". The I2RS protocol is a protocol designed to a higher level protocol comprised of a set of IETF existing protocols

(NETCONF [RFC6241], RESTCONF [RFC8044], and others) which have been extended to work together to support a new interface to the routing system. The I2RS protocol is a "reuse" management protocol which creates new management protocols by reusing existing protocols and extending these protocols for new uses, and has been designed to be implemented in phases [RFC7921].

This document suggests the following additions to Yang to support the I2RS control plane datastore. [I-D.ietf-i2rs-ephemeral-state] specifies the I2RS requirements for the ephemeral state.

Section 3 of this document defines optional additions to yang 1.1 to support I2RS ephemeral control plane datastore. The main addition is the datastore statement with four new substatements (dstype, ephemeral, protosup, validation). The protosup substatement has two valid substatements (protobase, protoadd). The validation substatement has has three new substatements: bulkchecks, caching, and nstransport.

Section 4 provides the augmentation to RFC7950 tables for these optional features.

2. Definitions

2.1. Requirements language

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

2.2. I2RS Definitions

The I2RS architecture [RFC7921] defines the following terms:

ephemeral data: is data which does not persist across a reboot (software or hardware) or a power on/off condition. Ephemeral data can be configured data or data recorded from operations of the router. Ephemeral configuration data also has the property that a system cannot roll back to a previous ephemeral configuration state. (See [RFC7921] for an architectural overview, [I-D.ietf-i2rs-ephemeral-state] for requirements, and [I-D.ietf-netmod-revised-datastores] for discussion of how the ephemeral datastore as a control plane datastore interacts with intended configuration datstore, the dynamic configuration protocols, and control planes datastore to create the applied datastore and operational state datastore.

local configuration: is the data on a routing system which does persist across a reboot (software or hardware) and a power on/off condition. Local configuration is defined as the intended datastore as defined in [I-D.ietf-netmod-revised-datastores].

dynamic configuration protocols datastore are configuration protocols such as DHCP that interact with the intended datastore (which does persist across a reboot (software or hardware) power on/off condition), and the I2RS ephemeral state control plane datastore.

applied datastore Read only datastore regarding configuration state installed in the routing system as defined in [I-D.ietf-netmod-revised-datastores].

operational state Read only datastore that combines applied datastore and operational state as defined in [I-D.ietf-netmod-revised-datastores].

operator-applied policy: is a policy that an operator sets that determines how the ephemeral datastore as a control plane data store interacts with intended configuration (see [I-D.ietf-netmod-revised-datastores]). This operator policy consists of setting a priority for each of the following (per [I-D.ietf-i2rs-ephemeral-state]):

- intended configuration,
- any dynamic configuration protocols,
- any control plane datastores (one of which is ephemeral.)

3. yang additions

3.1. datastoredef

The "datastoredef" is a statement that defines a datastore provides the ability to describe which datastore a module or submodule may be loaded into. Each datastore statement must refer to a name defined in a datastoredef statement.

The new substatements for the datstoredef command are dstype, ephemeral, module-list, precedence, protosup, and validation. The dstype provides information on the type of the datastore. The ephemeral flag indicates the datastore is ephemeral. The module-list provides a list of modules included in this datastore. the protosup indicates the protocol support for this datastore, and the validation provides information on the validation.

The "dsname" must be MUST be a nmae registered with IANA (see [I-D.ietf-netmod-revised-datastores]).

Data store syntax:

```
datastoredef <dsname> {
   <sub-statements>
};

dsname - Must be registered name for datastore
```

Figure 1

The substatements for the datastore substatement are listed below:

Table 1

substatement	This document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
dstype	3.3	-	1
ephemeral	3.4	-	0..n
module		7.1	0..n
module-list	3.5	-	0..n
organization	-	7.1.7	0..1
precedence	3.6	-	0..n
protosup	3.7	-	0..n
reference	-	7.21.4	0..1
revision	-	7.1.9	0..n
revision-date	-	7.1.5.1	0..1
validation	3.8	-	0..n
version		7.1.9	0..n

Note:There is a variance with ephemeral control-plane datastore example in [I-D.ietf-netmod-revised-datastores] which uses "module" to define a datastore rather than datastore. Rather than assume the "module" will be re-used this document uses "datastoredef".

Example of use:

```
datastoredef i2rs-agent {
         dstype control-plane;
     description {"I2RS Agent datastore "};
         ephemeral true;
         module-list ietf-i2rs-rib, ietf-network, ietf-network-topology,
          ietf-l3-unicast-topology;
         protosup {
                protobase netconf;
                protoadd control-plane;
        protoadd ephemeral;
         }
         precedence applied {
          precedenceval 5; //set to high value//
         }
     precedence opstate {
          precedence 5;   //set to high value//
         }
         revision 0.0;
         version 1.0;

}
```

3.2. datastore

The "datastore" can be a substatement for the Yang Module statement provides the ability to describe which datastore a module or submodule may be loaded into. If no "datastore" statement exists, there is no restriction on the datastores a module or submodule can be loaded into.

The argument the datastore is a datastore name denoted as "dsname". The "dsname" must be MUST be a nmae registered with IANA (see [I-D.ietf-netmod-revised-datastores]).

The vaid substatements for the datastore statement are in Table 1. The "description" substatement provides a description of the datastore. The "dstype" provides information on the class (e.g., config or control plane) and the subclass of the datastore (e.g., i2rs). The ephemeral indicates that entire datastore is ephemeral. The validation provides alternate validation rules for the datastore.

Data store syntax:

```
datastore <dsname> {
   <sub-statements>
};

dsname - must be defined in a datastoredef statement
```

Figure 2

The substatements for the datastore substatement are listed below:

Table 2

substatement	This document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
dstype	3.4	-	1
ephemeral	3.5	-	0..n
protosup	3.7	-	0..n
reference	-	7.21.4	0..1
revision	-	7.1.9	0..n
revision-date	-	7.1.5.1	0..1
validation	3.8	-	0..n
version		7.1.9	0..n

Application Comments:

A module may be mounted into different datastores. The datastore statement indicates which datastores a module may be mounted in, and the characteristics of each datastore.

Example of use where a module is utilized
in two different datastores.

```
   module ietf-i2rs-rib {
         yang-version 1.1;
        namespace "urn:ietf:params:xml:ns:yang:ietf-i2rs-rib";
        // replace with iana namespace when assigned
            prefix "iir";
            import ietf-inet-types {
               prefix inet;
                   //rfc6991
                   }
             ....
        organization
          "IETF I2RS (Interface to Routing System) Working Group";
             ....

         datastore i2rs-agent {
               dstype "control-plane" "i2rs-vo";
                   ephemeral true;
                   protosup {
                    protobase netconf;
                    protoadd control-plane;
                    protoadd ephemeral;
                    }
             }
             datastore config {
                   dstype config;
                   ephemeral false;
                   protosup {
                    protobase restconf;
                   }
                   protosup {
                    protobase netconf;
                    }
             }

   }
```

3.3. dstype

They substatement dstype indicates the datastore class and subclass of the datastore. A dstype substatement may only exist within a datastore statement.

```
   Syntax of the dstype datastore is:

   dstype <dsclass> <dsname>;

   where:
       dsclass: ["config" | "control-plane ]
       dssubclass [ "i2rs-v0" ]
```

Figure 3

3.4. ephemeral

The ephemeral indicates that an object is ephemeral data which does not survive a reboot (see [I-D.ietf-i2rs-ephemeral-state]). The definition of the object may be a datastore, a module, a submodule, an action, a container, a grouping, a leaf, a leaf-list, a list, or an rpc.

```
   Syntax is the following:
   ephemeral  [true | false];

   The value "true" indicates the object is not ephemeral.
   The value "false" indicates the value is ephemeral.
```

Figure 4

Note: There is a variance with ephemeral functionality with [I-D.ietf-netmod-revised-datastores]. Rather than consider the keyword ephemeral as a identity, this proposes ephemeral will be a yang substatement.

3.5. module-list

The module list contains a list of module names.

```
   Syntax is the following:
   module-list <module-name-1>, .... <module-name-n>;

   Each name on the list (e.g. <module-name-1>)
   must be a name in a module statement.
```

Figure 5

3.6. precedence

The precedence provides the value for precedence insertion of the datastores (the precedence substatement is contained) versus the datastore "dsname". Examples of datastore can be applied, opstate, or other control plane datastores. If no precedence is statement is given, the configuration datastore takes precedence.

The module-list restricts this precedence for just these modules. A submodule must belong to one of the modules in the module list, and it further restricts the precedence value to just the submodule within the module.

Syntax is the following:

```
precedence [applied | opstate | <dsname> ] {
    <<precedence-substatements>>
};

dsname - registered name for datastore
```

Figure 6

Table 3

substatement	document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
module-list	3.x	-	0..n
precedenceval	3.x	-	1
sub-modules		7.2	0..n

3.6.1. precedenceval

The precedenceval provides the value for precedence.

Syntax is the following:

```
precedenceval <precedence-integer>;

<precedence-integer>; - is the integer value for precedence.
```

Figure 7

3.7. protosup statement

This indicates which protocols support this datastore. The protocols can be netconf, restconf, coap, gprc, and bgp. The substatements for protosup are protocobase and protoadd

```
Syntax is the following:
protosup  {
    <<protosup substatements>>
}
```

Figure 8

Table 4

substatement	document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
protobase	3.4.1	-	1..n
protoadd	3.4.2	-	1..n

3.7.1. protobase

The protobase substatement indicates the protocol a database can be sent over. The syntax is below:

```
Syntax for protobase:

   protobase  [netconf | restconf | coap
               | bgp | isis | ospf | dots
                           | <protocol-name> ]


Where protocolo-name is one of protocol names
      registered by IANA.
```

Figure 9

3.7.2. protoadd

The protoadd specifies required optional additions to a protocol that sends information to a datastore. One example of such an addition is the additions to RESTCONF to support the I2RS protocol denoted as "i2rs".

Syntax for proto add:

```
   protoadd  [control-plane | i2rs | i2nsf |
             | ephemeral | <proto-add-string>]
```

Figure 10

The protocol additions is the name of a capability or grouping of capabilities for support. For example, the "i2rs" capability is a capability which combines the capabilities the "control-plane" netconf capability with the netconf ephemeral capability.

3.8. validation

The validation keyword indicates that the object uses alternate validation besides the mechanisms defined by the configuration datastore as defined in [RFC7950]. The validation subcommand is invalid in any module or submodule which is only defined for the configuration datastore. Unless the module has a datastore statement which includes a datastore other than config, all validation statements in the module are ignored. Unless the submodule has a datastore statement which includes a datastore other than config, all validation statements are ignored.

The validation can be set on a datastore command in a a module, a submodule, an action, a container, a grouping, a leaf, a leaf-list, a list, or an rpc. The validation substatements include nstransport and bulk-checks as shown in table 3.

Syntax of the validation is:

```
 validation {
    <<validation-substatements>>
   };
```

Figure 11

Table 5

substatement	document section	RFC7960 section	cardinality
description	-	7.21.3	0..1
bulkchecks	3.8.1	-	0..1
caching	3.8.2	-	0..1
nstransport	3.8.3	-	0..1
organization	-	7.1.7	0..1
reference	-	7.21.4	0..n
revision-date	-	7.1.5.1	0..1
version		7.1.9	0..n

3.8.1. bulkcheck

The bulkcheck flag indicates whether this object uses bulk-check validation instead of the normal configuration datastore validation. The protocol updating the object must support bulk checking mechanism, or indicate that this object is not supported.

This is a new feature for control plane protocols and control plane datastores. In the configuration datastores, it is possible to support this feature at the validation level for the rpc object. Early implemementers of this feature for module which may loaded in the configuration datastore are encouraged to place bulkcheck features within "rpc" functionality.

bulkcheck syntax:

```
bulkchecks  [yes | no];
```

Figure 12

The value "no" indicates the object does not allows "bulkchecks" of data, and uses the normal configuration datastore checking. The value "yes" indicates the object does not allows "bulkchecks" of data within this object.

3.8.2. caching

The caching flag indicates whether the I2RS support caching of multiple client information within I2RS Agents.

Application note: This feature is not supported for the I2RS protocol version 0 caching syntax:

```
caching  [yes | no];
```

Figure 13

The value "no" indicates the object does not allows "bulkchecks" of data, and uses the normal configuration datastore checking. The value "yes" indicates the object does not allows "bulkchecks" of data within this object.

3.8.3. nstransport

The nstransport indicates whether this object may be sent across a non-secure transport. Sending data across a non-secure transport should be done only if the circumstances warrant it.

This is a new feature for the I2RS control plane protocols and control plane datastores.

Caution: For a description of when a on-secure transport is appropriate for I2RS control plane protocol, please refer to the I2RS protocol security requirements [I-D.ietf-i2rs-protocol-security-requirements]. Implementers of this feature in an I2RS implementation should also review the I2RS security requirements [I-D.ietf-i2rs-security-environment-reqs]. No data which reveals any identity for a person or confidential information should be sent via a non-secure transport.

Syntax is the following:

```
nstransport  [yes | no];
```

Figure 14

The value "no" indicates the object does not allows "bulkchecks" of data, and uses the normal configuration datastore checking. The value "yes" indicates the object does not allows "bulkchecks" of data within this object.

4. Change to RFC7950

The optional attributes add options to the tables for substatements for the module (section 7.1.1), submodule table, action, container, grouping, leaf, leaf-list, a list, and an rpc. This section provides the revised tables.

4.1. Additions to the Module table

This is the additions to module’s substatment table in section 7.1.1 of [RFC7950].

7.1.1 substatement (replacement)

substatement	RFC7950 section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
namespace	7.1.3	1
notification	7.16	0..n
organization	7.1.7	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1
optional Yang 1.1 substatement	**This document’s section**	**cardinality**
datastore	3.2	0..n
ephemeral	3.4	0..n
validation	3.8	0..n

4.2. Additions to the submodule substatement list

Below would be the replacement for the substatement table in setion 7.2.1 of [RFC7950] which lists the valid submodule statements.

7.2.1. The submodule’s Substatements (replcement)

substatement	RFC7950 section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
belongs-to	7.2.2	1
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
namespace	7.1.3	1
notification	7.16	0..n
organization	7.1.7	0..1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1

optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.3. Additions to the container substatement list

Below would be the replacement for the substatement table in section 7.5.2 of [RFC7950] that lists the legal container substatements.

7.5.2. The container Substatements (replacement)

substatement	RFC7950 section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
notification	7.16	0..n
presennce	7.5.5	0..1
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n
uses	7.13	0..n
when	7.21.5	0..1

optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.4. Additions to leaf substatement list

Below would be replacement for the substatement table in section 7.6.2 of [RFC7950] which provides the leaf’s substatements.

7.6.2 The leaf’s Substatements (replacement)

substatement	RFC7950 section	cardinality
config	7.21.1	0..1
default	7.6.4	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.21.5	0..1

optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.5. Additions to leaf-list substatement list

Below would be the replacement for the substatement table in section 7.7.2 in [RFC7950] which provides the list of the leaf-lists substatements.

7.7.2 The leaf’s Substatements (replacement)

substatement	RFC7950 section	cardinality
config	7.21.1	0..1
default	7.6.4	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.6.3	1
units	7.3.3	0..1
when	7.21.5	0..1

optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.6. Additions to list substatement list

Below would be the replacement for the table in section 7.8.1 in [RFC7950] which provides the list’s substatements.

7.8.1 The list’s Substatements (replacement)

substatement	RFC7950 section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
key	7.8.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
max-elements	7.7.6	0..1
min-elements	7.7.5	0..1
must	7.5.3	0..n
notification	7.16	0..n
ordered-by	7.7.7	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n
when	7.21.5	0..1

optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.7. Additions to the grouping substatement table

Below would be the replacement for the table 7.12.1 of [RFC7950] that lists the vaid substatments for the container substatements.

7.12.1. The grouping’s Substatements (replacement)

substatement	RFC7950 section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
description	7.21.3	0..1
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n
uses	7.13	0..n
optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.8. Additions to the rpc substatement list

Below would be the replacement for the table in section 7.14.1 of [RFC7950] that lists the legal rpc substatements.

7.5.2. The rpc Substatements

substatement	RFC7950 section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n

optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

4.9. Additions to the action substatement list

Below would be the replacement for the table 7.15.1 of [RFC7950] that lists the legal action substatements.

7.5.2. The action Substatements

substatement	RFC7950 section	cardinality
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
input	7.14.2	0..1
output	7.14.3	0..1
reference	7.21.4	0..1
status	7.1.9	0..1
typedef	7.3	0..n

optional Yang 1.1 substatement	This document’s section	cardinality
ephemeral	3.4	0..n
validation	3.8	0..n

Figure 2

5. IANA Considerations

The additions for registries go here.

6. 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 these yang additions for an I2RS protocol should consider both security requirements.

7. Acknowledgements

tBD

8. References

8.1. Normative References:

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

[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>.

[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, <http://www.rfc-editor.org/info/rfc7921>.

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

[RFC8044] DeKok, A., "Data Types in RADIUS", RFC 8044, DOI 10.17487/RFC8044, January 2017, <http://www.rfc-editor.org/info/rfc8044>.

8.2. Informative References

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-23 (work in progress), November 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-17 (work in progress), September 2016.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-03 (work in progress), March 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", draft-ietf-netmod-revised-datastores-00 (work in progress), December 2016.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 11, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio

February 07, 2018

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-06

Abstract

This document defines encoding rules for serializing configuration data, state data, RPC input and RPC output, Action input, Action output and notifications defined within YANG modules using the Concise Binary Object Representation (CBOR) [RFC7049].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The specification of the YANG 1.1 data modelling language [RFC7950] defines an XML encoding for data instances, i.e. contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC7159]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [RFC7951].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC7049]. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- action

- anydata

- anyxml

- data node

- data tree

- datastore

- feature

- identity
- module
- notification
- RPC
- schema node
- schema tree
- submodule

The following terms are defined in [RFC7951]:

- member name
- name of an identity
- namespace-qualified

The following terms are defined in [RFC8040]:

- yang-data (YANG extension)
- YANG data template

This specification also makes use of the following terminology:

- child: A schema node defined within a collection such as a container, a list, a case, a notification, an RPC input, an RPC output, an action input, an action output.
- delta: Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- parent: The collection in which a schema node is defined.
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

2.1. YANG Schema Item iDentifier (SID)

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called YANG Schema Item iDentifier (SID), is encoded using an unsigned integer. The following items are identified using SIDs:

- identities
- data nodes
- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information
- YANG modules, submodules and features

To minimize its size, in certain positions, SIDs are represented using a (signed) delta from a reference SID and the current SID. Conversion from SIDs to deltas and back to SIDs are stateless processes solely based on the data serialized or deserialized.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness is outside the scope of the present specification. If SIDs are to be used, the present specification is used in conjunction with a specification defining this management. One example for such a specification is under development as [I-D.ietf-core-sid].

2.2. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

CBOR content	CBOR type	Diagnostic notation	Example	CBOR encoding
Unsigned integer	0	Decimal digits	123	18 7b
Negative integer	1	Decimal digits prefixed by a minus sign	-123	38 7a
Byte string	2	Hexadecimal value enclosed between single quotes and prefixed by an ’h’	h’f15c’	42 f15c
Text string	3	String of Unicode characters enclosed between double quotes	"txt"	63 747874
Array	4	Comma-separated list of values within square brackets	[1, 2]	82 01 02
Map	5	Comma-separated list of key : value pairs within curly braces	{ 1: 123, 2: 456 }	a2 01187b 021901c8
Boolean	7/20 7/21	false true	false true	f4 f5
Null	7/22	null	null	f6
Not assigned	7/23	undefined	undefined	f7

Table 1: CBOR diagnostic notation summary

The following extensions to the CBOR diagnostic notation are supported:

- Any text within and including a pair of slashes is considered a comment.

- Deltas are visualized as numbers preceded by a ’+’ or ’-’ sign. The use of the ’+’ sign for positive deltas represents an extension to the CBOR diagnostic notation as defined by [RFC7049] section 6.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG schema trees and their subtrees.

Basic schema nodes such as leaf, leaf-list, list, anydata and anyxml can be encoded standalone. In this case, only the value of this schema node is encoded in CBOR. Identification of this value needs to be provided by some external means when required.

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using a CBOR map in which each child schema node is encoded using a key and a value. This specification supports two type of CBOR keys; YANG Schema Item iDentifier (SID) as defined in Section 2.1 and member names as defined in [RFC7951]. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. The end user of this mapping specification (e.g. RESTCONF [RFC8040], CoMI [I-D.ietf-core-comi]) can mandate the use of a specific key type.

In order to minimize the size of the encoded data, the proposed mapping avoids any unnecessary meta-information beyond those natively supported by CBOR. For instance, CBOR tags are used solely in the case of anyxml data nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

4. Encoding of YANG Data Node Instances

Schema node instances defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC7049].

4.1. The ’leaf’ Data Node

Leafs MUST be encoded based on the encoding rules specified in Section 6.

4.2. The ’container’ Data Node

Collections such as containers, list instances, notifications, RPC inputs, RPC outputs, action inputs and action outputs MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. Each key within the CBOR map is set to a data node identifier, each value is set to the value of this data node instance according to the instance datatype.

This specification supports two type of CBOR keys; SID as defined in Section 2.1 encoded as deltas and member names as defined in [RFC7951] encoded using CBOR text strings. The use of CBOR byte strings for keys is reserved for future extensions.

4.2.1. SIDs as keys

Keys implemented using SIDs MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Keys are represented as the delta of the associated SID, delta values are computed as follows:

- The delta value is equal to the SID of the current schema node minus the SID of the parent schema node. When no parent exists in the context of use of this container, the delta is set to the SID of the current schema node (i.e., a parent with SID equal to zero is assumed).
- Delta values may result in a negative number, clients and servers MUST support both unsigned and negative deltas.

The following example shows the encoding of a 'system-state' container instance with a single child, a clock container. The clock container container has two children, a 'current-datetime' leaf and a 'boot-datetime' leaf.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
             \d{2}:\d{2})';
  }
}

container system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

    leaf boot-datetime {
      type date-and-time;
    }
  }
}
```

For this first representation, we assume that the SID of the parent container (i.e. 'system-state') is not available to the serializer. In this case, root data nodes are encoded using absolute SIDs.

CBOR diagnostic notation:

```
{
  1717 : {                             / clock  (SID 1717) /
    +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1719)/
    +1 : "2015-09-15T09:12:58Z-05:00"  / boot-datetime (SID 1718) /
  }
}
```

CBOR encoding:

```
a1                                       # map(1)
   19 06b5                               # unsigned(1717)
   a2                                    # map(2)
      02                                 # unsigned(2)
      78 1a                              # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
      01                                 # unsigned(1)
      78 1a                              # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030
```

On the other hand, if the serializer is aware of the parent SID, 1716 in the case 'system-state' container, root data nodes are encoded using deltas.

CBOR diagnostic notation:

```
{
  +1 : {                               / clock  (SID 1717) /
    +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1719)/
    +1 : "2015-09-15T09:12:58Z-05:00"  / boot-datetime (SID 1718) /
  }
}
```

CBOR encoding:

```
a1                                       # map(1)
   01                                     # unsigned(1)
   a2                                    # map(2)
      02                                 # unsigned(2)
      78 1a                              # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
      01                                 # unsigned(1)
      78 1a                              # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.2.2. Member names as keys

Keys implemented using member names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified member name MUST be used for all members of a top-level collection, and then also whenever the namespaces of the schema node and its parent are different. In all other cases, the simple form of the member name MUST be used. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of a 'system' container instance using names. This example is described in Section 4.2.1.

CBOR diagnostic notation:

```
{
  "ietf-system:clock" : {
    "current-datetime" : "2015-10-02T14:47:24Z-05:00",
    "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
  }
}
```

CBOR encoding:

```
a1                                        # map(1)
   71                                     # text(17)
      696574662d73797374656d3a636c6f636b  # "ietf-system:clock"
   a2                                     # map(2)
      70                                  # text(16)
         63757272656e742d6461746574696d65 # "current-datetime"
      78 1a                               # text(26)
         323031352d31302d30325431343a34373a32345a2d30353a3030
      6d                                  # text(13)
         626f6f742d6461746574696d65       # "boot-datetime"
      78 1a                               # text(26)
         323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.3. The 'leaf-list' Data Node

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array MUST be encoded using the rules defined by the YANG type specified.

The following example shows the encoding a 'search' leaf-list instance containing the two entries, "ietf.org" and "ieee.org".

Definition example [RFC7317]:

```
typedef domain-name {
  type string {
    length "1..253";
    pattern '((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9].)
             *([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.?
             )|\.';
  }
}

leaf-list search {
  type domain-name;
  ordered-by user;
}
```

CBOR diagnostic notation: ["ietf.org", "ieee.org"]

CBOR encoding: 82 68 696574662e6f7267 68 696565652e6f7267

4.4. The 'list' Data Node

A list MUST be encoded using a CBOR array data item (major type 4). Each list instance within this CBOR array is encoded using a CBOR map data item (major type 5) based on the same rules as a YANG container as defined in Section 4.2.

4.4.1. SIDs as keys

The following example show the encoding of a 'server' list instance using SIDs. It is important to note that the protocol or method using this mapping may carry a parent SID or may have the knowledge of this parent SID based on its context. In these cases, delta encoding can be performed based on this parent SID which minimizes the size of the encoded data.

Definition example from [RFC7317]:

```
   list server {
     key name;

     leaf name {
       type string;
     }
     choice transport {
       case udp {
         container udp {
           leaf address {
             type host;
             mandatory true;
           }
           leaf port {
             type port-number;
           }
         }
       }
     }
     leaf association-type {
       type enumeration {
         enum server;
         enum peer;
         enum pool;
       }
       default server;
     }
     leaf iburst {
       type boolean;
       default false;
     }
     leaf prefer {
       type boolean;
       default false;
     }
   }
```

CBOR diagnostic notation:

```
[
  {
    1755 : "NRC TIC server",          / name (SID 1755) /
    1757 : {                          / udp (SID 1757) /
      +1 : "tic.nrc.ca",              / address (SID 1758) /
      +2 : 123                        / port (SID 1759) /
    },
    1753 : 0,                         / association-type (SID 1753) /
    1754 : false,                     / iburst (SID 1754) /
    1756 : true                       / prefer (SID 1756) /
  },
  {
    1755 : "NRC TAC server",          / name (SID 1755) /
    1757 : {                          / udp (SID 1757) /
      +1 : "tac.nrc.ca"               / address (SID 1758) /
    }
  }
]
```

CBOR encoding:

```
82                                    # array(2)
   a5                                 # map(5)
      19 06db                         # unsigned(1755)
      6e                              # text(14)
         4e52432054494320736572766572 # "NRC TIC server"
      19 06dd                         # unsigned(1757)
      a2                              # map(2)
         01                           # unsigned(1)
         6a                           # text(10)
            7469632e6e72632e6361      # "tic.nrc.ca"
         02                           # unsigned(2)
         18 7b                        # unsigned(123)
      19 06d9                         # unsigned(1753)
      00                              # unsigned(0)
      19 06da                         # unsigned(1754)
      f4                              # primitive(20)
      19 06dc                         # unsigned(1756)
      f5                              # primitive(21)
   a2                                 # map(2)
      19 06db                         # unsigned(1755)
      6e                              # text(14)
         4e52432054414320736572766572 # "NRC TAC server"
      19 06dd                         # unsigned(1757)
      a1                              # map(1)
         01                           # unsigned(1)
         6a                           # text(10)
            7461632e6e72632e6361      # "tac.nrc.ca"
```

4.4.2. Member names as keys

The following example shows the encoding of a ’server’ list instance using names. This example is described in Section 4.4.1.

CBOR diagnostic notation:

```
[
  {
    "ietf-system:name" : "NRC TIC server",
    "ietf-system:udp" : {
      "address" : "tic.nrc.ca",
      "port" : 123
    },
    "ietf-system:association-type" : 0,
    "ietf-system:iburst" : false,
    "ietf-system:prefer" : true
  },
  {
    "ietf-system:name" : "NRC TAC server",
    "ietf-system:udp" : {
      "address" : "tac.nrc.ca"
    }
  }
]
```

CBOR encoding:

```
   82                                           # array(2)
      a5                                        # map(5)
         70                                     # text(16)
            696574662d73797374656d3a6e616d65    # "ietf-system:name"
         6e                                     # text(14)
            4e524320544943207365727665 72       # "NRC TIC server"
         6f                                     # text(15)
            696574662d73797374656d3a756470      # "ietf-system:udp"
         a2                                     # map(2)
            67                                  # text(7)
               61646472657373                   # "address"
            6a                                  # text(10)
               7469632e6e72632e6361             # "tic.nrc.ca"
            64                                  # text(4)
               706f7274                         # "port"
            18 7b                               # unsigned(123)
         78 1c                                  # text(28)
            696574662d73797374656d3a6173736f63696174696f6e2d74797065
         00                                     # unsigned(0)
         72                                     # text(18)
            696574662d73797374656d3a696275727374 # "ietf-system:iburst"
         f4                                     # primitive(20)
         72                                     # text(18)
            696574662d73797374656d3a707265666572 # "ietf-system:prefer"
         f5                                     # primitive(21)
      a2                                        # map(2)
         70                                     # text(16)
            696574662d73797374656d3a6e616d65    # "ietf-system:name"
         6e                                     # text(14)
            4e52432054414320736572766572        # "NRC TAC server"
         6f                                     # text(15)
            696574662d73797374656d3a756470      # "ietf-system:udp"
         a1                                     # map(1)
            67                                  # text(7)
               61646472657373                   # "address"
            6a                                  # text(10)
               7461632e6e72632e6361             # "tac.nrc.ca"
```

4.5. The ’anydata’ Data Node

An anydata serves as a container for an arbitrary set of schema nodes that otherwise appear as normal YANG-modeled data. An anydata instance is encoded using the same rules as a container, i.e., CBOR map. The requirement that anydata content can be modeled by YANG implies the following:

- o Keys of any inner data nodes MUST be set to valid deltas or member names.

- The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).

- Values MUST follow the encoding rules of one of the datatypes listed in Section 6.

The following example shows a possible use of anydata. In this example, an anydata is used to define a data node containing a notification event, this data node can be part of a YANG list to create an event logger.

Definition example:

```
anydata event;
```

This example also assumes the assistance of the following notification.

```
module example-port {
  ...

  notification example-port-fault {  # SID 2600
    leaf port-name {                 # SID 2601
      type string;
    }
    leaf port-fault {                # SID 2601
      type string;
    }
  }
}
```

CBOR diagnostic notation:

```
{
  2601 : "0/4/21",       / port-name /
  2602 : "Open pin 2"    / port-fault /
}
```

CBOR encoding:

```
a2                         # map(2)
   19 0a29                 # unsigned(2601)
   66                      # text(6)
      302f342f3231         # "0/4/21"
   19 0a2a                 # unsigned(2602)
   6a                      # text(10)
      4f70656e2070696e2032 # "Open pin 2"
```

4.6. The ’anyxml’ Data Node

An anyxml schema node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. anyxml value may contain CBOR data items tagged with one of the tag listed in Section 8.1, these tags shall be supported.

The following example shows a valid CBOR encoded instance.

Definition example from [RFC7951]:

```
anyxml bar;
```

CBOR diagnostic notation: [true, null, true]

CBOR encoding: 83 f5 f6 f5

5. Encoding of YANG data templates

YANG data templates are data structures defined in YANG but not intended to be implemented as part of a datastore. YANG data templates are defined using the ’yang-data’ extension as described by RFC 8040.

The encoding rules defined for YANG containers in section 4.2 may be used to serialize YANG data templates.

Definition example from [I-D.ietf-core-comi]:

```
   import ietf-restconf {
     prefix rc;
   }

   rc:yang-data yang-errors {
     container error {
       leaf error-tag {
         type identityref {
           base error-tag;
         }
       }
       leaf error-app-tag {
         type identityref {
           base error-app-tag;
         }
       }
       leaf error-data-node {
         type instance-identifier;
       }
       leaf error-message {
         type string;
       }
     }
   }
```

Just like YANG containers, YANG data templates can be encoded using either SIDs or names.

5.1. SIDs as keys

This example shows a serialization example of the yang-errors template using SIDs as CBOR map key.

CBOR diagnostic notation:

```
   {
     1024 : {                    / error  (SID 1024) /
       +4 : 1011,                / error-tag (SID 1028) /
                                 / = invalid-value (SID 1011) /
       +1 : 1018,                / error-app-tag (SID 1025) /
                                 / = not-in-range (SID 1018) /
       +2 : 1740,                / error-data-node (SID 1026) /
                                 / = timezone-utc-offset (SID 1740) /
       +3 : "max value exceeded" / error-message (SID 1027) /
     }
   }
```

CBOR encoding:

```
A1                                      # map(1)
   19 0400                              # unsigned(1024)
   A4                                   # map(4)
      04                                # unsigned(4)
      19 03F3                           # unsigned(1011)
      01                                # unsigned(1)
      19 03FA                           # unsigned(1018)
      02                                # unsigned(2)
      19 06CC                           # unsigned(1740)
      03                                # unsigned(3)
      76                                # text(22)
         6D6178696D756D2076616C7565206578636565646564
```

5.2. Member names as keys

This example shows a serialization example of the yang-errors template using member names as CBOR map key.

CBOR diagnostic notation:

```
{
  "ietf-comi:error" : {
    "error-tag" : "invalid-value",
    "error-app-tag" : "not-in-range",
    "error-data-node" : "timezone-utc-offset",
    "error-message" : "max value exceeded"
  }
}
```

CBOR encoding:

```
   A1                                      # map(1)
      6F                                   # text(15)
         696574662D636F6D693A6572726F72
      A4                                   # map(4)
         69                                # text(9)
            6572726F722D746167             # "error-tag"
         6D                                # text(13)
            696E76616C69642D76616C7565
         6D                                # text(13)
            6572726F722D6170702D746167
         6C                                # text(12)
            6E6F742D696E2D72616E6765
         6F                                # text(15)
            6572726F722D646174612D6E6F6465
         73                                # text(19)
            74696D657A6F6E652D7574632D6F6666736574
         6D                                # text(13)
            6572726F722D6D657373616765
         72                                # text(18)
            6D61782076616C7565206578636565646564
```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list data node depends on the built-in type of that data node. The following sub-section defined the CBOR encoding of each built-in type supported by YANG as listed in [RFC7950] section 4.2.4. Each subsection shows an example value assigned to a data node instance of the discussed built-in type.

6.1. The unsigned integer Types

Leafs of type uint8, uint16, uint32 and uint64 MUST be encoded using a CBOR unsigned integer data item (major type 0).

The following example shows the encoding of a ’mtu’ leaf instance set to 1280 bytes.

Definition example from [RFC7277]:

```
leaf mtu {
  type uint16 {
    range "68..max";
  }
}
```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

6.2. The integer Types

Leafs of type int8, int16, int32 and int64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value.

The following example shows the encoding of a 'timezone-utc-offset' leaf instance set to -300 minutes.

Definition example from [RFC7317]:

```
leaf timezone-utc-offset {
  type int16 {
    range "-1500 .. 1500";
  }
}
```

CBOR diagnostic notation: -300

CBOR encoding: 39 012b

6.3. The 'decimal64' Type

Leafs of type decimal64 MUST be encoded using a decimal fraction as defined in [RFC7049] section 2.4.3.

The following example shows the encoding of a 'my-decimal' leaf instance set to 2.57.

Definition example from [RFC7317]:

```
leaf my-decimal {
  type decimal64 {
    fraction-digits 2;
    range "1 .. 3.14 | 10 | 20..max";
  }
}
```

CBOR diagnostic notation: 4([-2, 257])

CBOR encoding: c4 82 21 19 0101

6.4. The ’string’ Type

Leafs of type string MUST be encoded using a CBOR text string data item (major type 3).

The following example shows the encoding of a ’name’ leaf instance set to "eth0".

Definition example from [RFC7223]:

```
leaf name {
  type string;
}
```

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

6.5. The ’boolean’ Type

Leafs of type boolean MUST be encoded using a CBOR true (major type 7, additional information 21) or false data item (major type 7, additional information 20).

The following example shows the encoding of an ’enabled’ leaf instance set to ’true’.

Definition example from [RFC7317]:

```
leaf enabled {
  type boolean;
}
```

CBOR diagnostic notation: true

CBOR encoding: f5

6.6. The ’enumeration’ Type

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Enumeration values are either explicitly assigned using the YANG statement ’value’ or automatically assigned based on the algorithm defined in [RFC7950] section 9.6.4.2.

The following example shows the encoding of an ’oper-status’ leaf instance set to ’testing’.

Definition example from [RFC7317]:

```
leaf oper-status {
  type enumeration {
    enum up { value 1; }
    enum down { value 2; }
    enum testing { value 3; }
    enum unknown { value 4; }
    enum dormant { value 5; }
    enum not-present { value 6; }
    enum lower-layer-down { value 7; }
  }
}
```

CBOR diagnostic notation: 3

CBOR encoding: 03

6.7. The ’bits’ Type

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 2). Bits position are either explicitly assigned using the YANG statement ’position’ or automatically assigned based on the algorithm defined in [RFC7950] section 9.7.4.2.

Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are assigned similarly. Within each byte, bits are assigned from least to most significant.

The following example shows the encoding of a ’mybits’ leaf instance with the ’disable-nagle’ and ’10-Mb-only’ flags set.

Definition example from [RFC7950]:

```
leaf mybits {
  type bits {
    bit disable-nagle {
      position 0;
    }
    bit auto-sense-speed {
      position 1;
    }
    bit 10-Mb-only {
      position 2;
    }
  }
}
```

CBOR diagnostic notation: h’05’

CBOR encoding: 41 05

6.8. The ’binary’ Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

The following example shows the encoding of an ’aes128-key’ leaf instance set to 0x1f1ce6a3f42660d888d92a4d8030476e.

Definition example:

```
leaf aes128-key {
  type binary {
    length 16;
  }
}
```

CBOR diagnostic notation: h’1f1ce6a3f42660d888d92a4d8030476e’

CBOR encoding: 50 1f1ce6a3f42660d888d92a4d8030476e

6.9. The ’leafref’ Type

Leafs of type leafref MUST be encoded using the rules of the schema node referenced by the ’path’ YANG statement.

The following example shows the encoding of an ’interface-state-ref’ leaf instance set to "eth1".

Definition example from [RFC7223]:

```
typedef interface-state-ref {
  type leafref {
    path "/interfaces-state/interface/name";
  }
}

container interfaces-state {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf-list higher-layer-if {
      type interface-state-ref;
    }
  }
}
```

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

6.10. The ’identityref’ Type

This specification supports two approaches for encoding identityref, a YANG Schema Item iDentifier (SID) as defined in Section 2.1 or a name as defined in [RFC7951] section 6.8.

6.10.1. SIDs as identityref

When schema nodes of type identityref are implemented using SIDs, they MUST be encoded using a CBOR unsigned integer data item (major type 0). (Note that no delta mechanism is employed for SIDs as identityref.)

The following example shows the encoding of a ’type’ leaf instance set to the value ’iana-if-type:ethernetCsmacd’ (SID 1180).

Definition example from [RFC7317]:

```
identity interface-type {
}

identity iana-interface-type {
  base interface-type;
}

identity ethernetCsmacd {
  base iana-interface-type;
}

leaf type {
  type identityref {
    base interface-type;
  }
}
```

CBOR diagnostic notation: 1180

CBOR encoding: 19 049c

6.10.2. Name as identityref

Alternatively, an identityref may be encoded using a name as defined in [RFC7951] section 6.8. When names are used, identityref MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in another module than the leaf node containing the identityref value, the namespace-qualified form MUST be used. Otherwise, both the simple and namespace-qualified forms are permitted. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its name. This example is described in Section 6.10.1.

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b
69616e612d69662d747970653a657468657726e657443736d616364

6.11. The 'empty' Type

Leafs of type empty MUST be encoded using the CBOR null value (major type 7, additional information 22).

The following example shows the encoding of a 'is-router' leaf instance when present.

Definition example from [RFC7277]:

```
leaf is-router {
  type empty;
}
```

CBOR diagnostic notation: null

CBOR encoding: f6

6.12. The 'union' Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed. When used in a union, the following YANG datatypes are prefixed by CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- bits
- enumeration
- identityref
- instance-identifier

See Section 8.1 for more information about these CBOR tags.

The following example shows the encoding of an 'ip-address' leaf instance when set to "2001:db8:a0b:12f0::1".

Definition example from [RFC7317]:

```
typedef ipv4-address {
  type string {
  pattern ’(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
           ([0-9][1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(%[\p{N}
           \p{L}]+)?’;
  }
}

typedef ipv6-address {
  type string {
    pattern ’((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-9a
             -fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0
             -9]|[01]?[0-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0
             -9]?[0-9])))(%[\p{N}\p{L}]+)?’;
    pattern ’(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|((([^:]+:)*[^:]+)
             ?::(([^:]+:)*[^:]+)?)(%.+)?’;
  }
}

typedef ip-address {
  type union {
    type ipv4-address;
    type ipv6-address;
  }
}

leaf address {
  type inet:ip-address;
}
```

CBOR diagnostic notation: "2001:db8:a0b:12f0::1"

CBOR encoding: 74 323030313a6462383a6130623a313266303a3a31

6.13. The ’instance-identifier’ Type

This specification supports two approaches for encoding an instance-identifier, one based on YANG Schema Item iDentifier (SID) as defined in Section 2.1 and one based on names as defined in [RFC7951] section 6.11.

6.13.1. SIDs as instance-identifier

SIDs uniquely identify a data node. In the case of a single instance data node, a data node defined at the root of a YANG module or submodule or data nodes defined within a container, the SID is sufficient to identify this instance.

In the case of a data node member of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Single instance data nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

Data nodes member of a YANG list MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.
- The following entries MUST contain the value of each key required to identify the instance of the targeted data node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from top level list, and follow by each of the subordinate list(s).

First example:

The following example shows the encoding of a leaf instance of type instance-identifier which identifies the data node "/system/contact" (SID 1737).

Definition example from [RFC7317]:

```
container system {

  leaf contact {
    type string;
  }

  leaf hostname {
    type inet:domain-name;
  }
}
```

CBOR diagnostic notation: 1737

CBOR encoding: 19 06c9

Second example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the data node instance

"/system/authentication/user/authorized-key/key-data" (SID 1730) for user name "bob" and authorized-key "admin".

Definition example from [RFC7317]:

```
list user {
  key name;

  leaf name {
    type string;
  }
  leaf password {
    type ianach:crypt-hash;
  }

  list authorized-key {
    key name;

    leaf name {
      type string;
    }
    leaf algorithm {
      type string;
    }
    leaf key-data {
      type binary;
  }
}
```

CBOR diagnostic notation: [1730, "bob", "admin"]

CBOR encoding:

```
83                        # array(3)
   19 06c2                # unsigned(1730)
   63                     # text(3)
      626f62              # "bob"
   65                     # text(5)
      61646d696e          # "admin"
```

Third example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the list instance "/system/authentication/user" (SID 1726) corresponding to the user name "jack".

CBOR diagnostic notation: [1726, "jack"]

CBOR encoding:

```
82                        # array(2)
   19 06be                # unsigned(1726)
   64                     # text(4)
      6a61636b            # "jack"
```

6.13.2. Names as instance-identifier

The use of names as instance-identifier is defined in [RFC7951] section 6.11. The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

First example:

This example is described in Section 6.13.1.

CBOR diagnostic notation: "/ietf-system:system/contact"

CBOR encoding:

```
78 1c 2f20696574662d73797374656d3a73797374656d2f636f6e74616374
```

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name=’bob’]/authorized-key
[name=’admin’]/key-data"

CBOR encoding:

```
78 59
   2f696574662d73797374656d3a73797374656d2f61757468656e74696361
   74696f6e2f757365725b6e616d653d27626f62275d2f617574686f72697a
   65642d6b65795b6e616d653d2761646d696e275d2f6b65792d64617461
```

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name=’bob’]"

CBOR encoding:

```
78 33
   2f696574662d73797374656d3a73797374656d2f61757468656e74696361
   74696f6e2f757365725b6e616d653d27626f62275d
```

7. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, this encoding does not contribute any new security issues in addition of those identified for the specific protocol or context for which it is used.

To minimize security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

8. IANA Considerations

8.1. Tags Registry

This specification requires the assignment of CBOR tags for the following YANG datatypes. These tags are added to the Tags Registry as defined in section 7.2 of [RFC7049].

Tag	Data Item	Semantics	Reference
40	bits	YANG bits datatype	RFC XXXX
41	enumeration	YANG enumeration datatype	RFC XXXX
42	identityref	YANG identityref datatype	RFC XXXX
43	instance-identifier	YANG instance-identifier datatype	RFC XXXX

// RFC Ed.: update Tag values using allocated tags if needed and remove this note // RFC Ed.: replace XXXX with RFC number and remove this note

9. Acknowledgments

This document has been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.ietf-core-comi]. [RFC7951] has also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Juergen Schoenwaelder.

10. References

10.1. 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>.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[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>.

10.2. Informative References

[I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-02 (work in progress), December 2017.

[I-D.ietf-core-sid] Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-03 (work in progress), December 2017.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

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

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

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

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

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

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

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

Alexander Pelov (editor)
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: http://www.landisgyr.com/

Ana Minaburo
Acklio
2bis rue de la chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

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

March 13, 2017

Filter-Based RIB Data Model
draft-ietf-i2rs-fb-rib-data-model-01

Abstract

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

The Filter based RIB is a protocol independent data structure which can be deployed in a configuration datastore, an ephemeral control plane data stroe.

Status of This Memo

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document provides a protocol-independent yang module for Filter Based Routing (FB-RIB) routing filters within a routing element. The informational model for this FB-RIB is in [I-D.ietf-i2rs-fb-rib-info-model].

1.1. Definition of Filter Based RIB

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

A Filter-Based RIB (Routing Information Base) is contained in a routing instance. It contains a list of filters (match-action conditions) and a list of interfaces the filter-based forwarding operates on, and default RIB(s).

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

event = packet/frame received,

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

action - modify packet and forward/drop packet.

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

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

- created by the I2RS Routing Informational Base (RIB) manager using the yang model described in: in [I-D.ietf-i2rs-rib-info-model], or

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

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

The Filter-Based RIB can reside in the configuration datastore, a control plane datastore, or an ephemeral control plane data store (e.g. I2RS ephemeral control plane datastore).

The Interface to the Routing System (I2RS) [RFC7921] architecture provides dynamic read and write access to the information and state within the routing elements. The I2RS client interacts with the I2RS agent in one or more network routing systems. The I2RS architecture defines the I2RS control plane datastore as ephemeral - which means it does not persist across a reboot.

2. Requirements Language

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

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

3. Definitions and Acronyms

CLI

Command Line Interface

FB-RIB

Filter-Based Routing Information Base

FB-Route

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

Policy Group

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

RIB IM

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

Routing instance

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

4. High level Yang structure for the FB-RIB

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

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

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

This yang model allows for three types of FB-RIB installations in three types of datastores:

> configuration (Config=TRUE, ephemeral=false, opstate definitions)
>
> ephemeral control plane (E.g. I2RS Agent, config=TRUE, ephemeral=TRUE, opstate definitions), and
>
> non-ephemeral control plane datastore (e.g. dBGP FB-FIB with config=TRUE; ephemeral=false, opstate which stores BGP Flow Specification received by bgp speaker from BGP peers).

Each of these cases is differentiated by using an "if-feature" to provide unique RIB under the routing instance.

Configuration RIBS

```
      +-----------------------------------------+
      |     routing instance                    |
      +-------|-------------|----------------|--+
              |             |                |
              |             |                |
    +---------|----+  +-----|-----+ +--------|-----+
    |config-fb-rib |  |i2rs-fb-rib| |bgp-fs-fb-rib |
    |              |  |           | |              |
    +------|-------+  +-----|-----+ +------|-------+
           |............:...|................|
                        :  (uses common structures
                        :   in separate lists of FB-RIBs)
               +--------|----+
               |fb-ribs*     |
               |             |
               +--|----------+
                  |
```

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

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

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

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

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

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

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

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

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

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

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

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

Figure 5: configuration state

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

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

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

Figure 5: operational state

4.2. Filter-Based RIB structures

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

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

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

Figure 6: FB RIB Type Structure

Note: acls:access-lists is the list of ACL filters in [I-D.ietf-netmod-acl-model].

HIgh Level Yang

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

5. yang models

5.1. Filter-Based RIB types

```
 <CODE BEGINS> file "ietf-fb-rib-types@2017-03-13.yang"
  module ietf-fb-rib-types {

   yang-version "1";

   // namespace
     namespace "urn:ietf:params:xml:ns:yang:ietf-fb-rib-types";
```

```
     prefix "fb-rib-t";
         import ietf-interfaces {prefix "if";}
         import ietf-routing {prefix "rt";}
         import ietf-pkt-eca-policy {prefix "pkt-eca";}
         import ietf-access-control-lists {prefix "acls";}

   // meta
   organization
     "IETF";

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

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

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

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


   revision "2017-03-13" {
       description
         "Filter-Based RIB protocol ";
         reference "draft-ietf-i2rs-fb-rib-data-model-01";
     }

     typedef fb-rib-type-def {
            type identityref {
              base "fb-rib-type";
                  }
                 description
                 "This type is used to refer to
                  source of Filter-Based RIB:
```

```
                  configuration, I2RS, Flow-Spec.";
      }

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

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

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

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

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

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

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

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

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

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

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

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

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

      grouping fb-rib-group-order-status {
           leaf group-refcnt {
             type uint16;
```

```
             description "refcnt for this group";
                 }
            leaf group-installed {
             type uint32;
             description "number of rules installed";
                 }
                 leaf group-matches {
                  type uint64;
                  description "number of matches by all
                   rules in group";
                 }
                 description "fb-rib group list order
                    and status info.";
           }

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

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

```
                 }

                 leaf bgp-fs-rib-name {
                     type string;
                         description "name of BGP
                          flow specification default RIB";
                 }
             description "default RIB for forwarding
                    if the policy match";
     }

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

```
                        uses fb-rib-updates;  // write refs to this RIB
                 list instance-using {
                          key instance-name;
                          leaf instance-name {
                            type string;
                            description
                                " name of instance using this fb-rib
                                 rt:routing-instance";
                           }
                          description "instances using
                           this fb-rib";
                        }
                    // ordered rule list + group list
                    uses pkt-eca:pkt-eca-policy-set;

                        // ordered acl list
                        uses acls:access-lists;

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

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

```
              status";
     }


  }

 <CODE ENDS>
```

5.2. FB-RIB

```
 <CODE BEGINS> file "ietf-fb-rib@2017-03-13.yang"
module ietf-fb-rib {
  yang-version "1";

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


  // import some basic inet types
  import ietf-yang-types {prefix "yang";}
  import ietf-fb-rib-types { prefix "fb-rib-t";}

  // meta
  organization
    "IETF";

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

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

     revision "2017-03-13" {
       description "initial revision";
       reference "draft-ietf-i2rs-fb-rib-data-model-01";
     }

        feature config-filter-based-RIB {
```

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


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

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

```
         }

        container ietf-fb-rib-opstate {
           presence "top-level structure for
            op-state";
           config "false";
       leaf default-instance-name {
             type string;
                 mandatory true;
        description
          "A routing instance is identified by its name,
           INSTANCE_name.  This MUST be unique across all routing
           instances in a given network device.";
         }
             leaf default-router-id {
                    type yang:dotted-quad;
                    description "Default router id";
                }
                container config-fb-rib-opstate {
                  if-feature config-filter-based-RIB;
                  uses fb-rib-t:fb-ribs-oper-status;
                 description "config filter-based RIB";
                }
                container i2rs-fb-rib-opstate {
                  if-feature I2RS-filter-based-RIB;
                  uses fb-rib-t:fb-ribs-oper-status;
                 description "bgp-fs filter-based RIB";
                }
                container bgp-fs-fb-rib-opstate {
                  if-feature BGP-FS-filter-based-RIB;
                  uses fb-rib-t:fb-ribs-oper-status;
                 description "bgp fs filter-based RIB";
                }
           description "fb-rib augments routing instance";
         }
}


<CODE ENDS>
```

6. IANA Considerations

TBD

7. Security Considerations

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

8. References

8.1. Normative References:

[I-D.ietf-i2rs-pkt-eca-data-model]
Hares, S., Wu, Q., and R. White, "Filter-Based Packet Forwarding ECA Policy", draft-ietf-i2rs-pkt-eca-data-model-02 (work in progress), October 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-07 (work in progress), January 2017.

[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-10 (work in progress), March 2017.

[I-D.ietf-netmod-routing-cfg]
Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-25 (work in progress), November 2016.

8.2. Informative References

[I-D.ietf-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-ietf-i2rs-fb-rib-info-model-00 (work in progress), June 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-10 (work in progress), December 2016.

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

[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, <http://www.rfc-editor.org/info/rfc7921>.

Authors’ Addresses

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Sriganesh Kini
Ericsson

Email: sriganesh.kini@ericsson.com

Linda Dunbar
Huawei
USA

Email: linda.dunbar@huawei.com

Ram Krishnan
Dell

Email: Ramkri123@gmail.com

Dean Bogdanovic
Juniper Networks
Westford, MA

Email: ivandean@gmail.org

Russ White
Linkedin

Email: russ@riw.us

I2RS working group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

S. Hares
L. Dunbar
Huawei
R. White
Ericsson

March 13, 2017

Filter-Based Packet Forwarding ECA Policy
draft-ietf-i2rs-pkt-eca-data-model-03.txt

Abstract

This document describes the yang data model for packet forwarding policy that filters received packets and forwards (or drops) the packets. Filters for Layer 2, Layer 3, Layer 4, and packet-arrival time are linked together to support filtering for the routing layer. 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.

This data model may be used in either the configuration datastore, control plane datastores, or the I2RS ephemeral control plane datastore.

Status of This Memo

Copyright Notice

Table of Contents

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. Filters for Layer 2, Layer 3, Layer-4 and packet arrival time are linked together to support filtering for the routing layer.

If one considers the reception of a packet as an event, this minimalistic Event-Match Condition-Action policy. Full event-match-condition policy can be found at [I-D.ietf-supa-generic-policy-data-model] (or the information model at [I-D.ietf-supa-generic-policy-info-model]). This document will use the term packet-only ECA policy for this model utilizing the term "packet" in this fashion.

ACL data models [I-D.ietf-netmod-acl-model] can also provide a minimal set of filtering for packet-eca by compiling a large group of filters. However, this data model also provides the L2-L4 filters plus a concept of grouping and policy rules. The pkt-eca structure helps create users with structures with more substantial policy for security or data flow direction.

This packet-only ECA policy data model supports an ordered list of ECA policy rules

- packet headers for layer 2 to layer 4,
- interfaces the packet was received on,
- time packet was received, and
- size of packet.

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

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

The forwardinng 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).

This packet policy draft has been developed as a set of protocol independent policy It may be used for the configuration datastore, a control plane datastore, or an I2RS ephemeral control plane datastore [RFC7921]. For more information configuration and control plane datastores please see [I-D.ietf-netmod-revised-datastores]. This yang model may be transmitted over NETCONF [RFC6241] or RESTCONF [RFC8040]. For use with the control plane datastores and ephemeral control plane datastores, additional capabilities support control plane daatastores will need to be added to the base NETCONF and RESTCONF to support these datastores.

This yang data model depends on the the I2RS RIB [I-D.ietf-i2rs-rib-data-model] which can be deployed in an configuration datastore, a control plane datastore, or the I2RS ephemeral control plane datastore.)for informational module see [I-D.ietf-i2rs-rib-info-model]. The update of RIB entries via the rpc features allows datastore validation differences to be handled in the rpc code.

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

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.

```
       +-----------+     +------------+
       |Rule Group |     | Rule Group |
       +-----------+     +------------+
            ^                  ^
                    |                  |
            |                  |
   +--------^-------+   +-------^-------+
   |      Rule      |   |     Rule      |
   +----------------+   +---------------+
                          :          :
                          :          :
                    ......:          :.....
                    :                     :
          +---------V---------+         +-V-------------+
          |  Rule Condition   |         |  Rule Action  |
          +-------------------+         +---------------+
              :     :    :                 :     :    :
         .....:     .    :.....       .....:     .    :.....
         :          :         :       :          :         :
    +----V---+  +---V----+ +--V---+ +-V------++--V-----++--V---+
    |  Match |  | match  | |match | | Action || action ||action|
    |Operator|  |Variable| |Value | |Operator||Variable|| Value|
    +--------+  +--------+ +------+ +--------++--------++------+
```

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:

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

Rule groups may have multiple policy groups at specific orders. For example, policy gorup 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 as shown in figure 2. The order indicates the order of the rule within the 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
        +---------------------------------------+
        |             Rule Group                |
        +---------------------------------------+
          *      *                   *
          |      |                   |
          |      |                   |
          |      |                   |
     +------+   +-------------------+
     | Name |   |     Rule_list     |
     |      |   |                   |
     +------+   +------|------------+
      +----------------|-----------------------------+
      |           rule                               |
      |-|------|----------|-----------|------------|-+
        |      |          |           |            |
    +---|--+ +-|----+ +---|-------+ +-|------+ +-------+
    | Name | |rule  | | ECA       | |rule    | |ref-cnt|
    +------+ |order | | match     | |priority| +-------+
             |number| |qos-actions| +--------+
             +------+ |fwd-actions|
                      +-----------+
```

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 (L2, L3, or L4) or time of day or packet size. The qos actions can be setting fields in the packet at any layer (L2-l4) 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
                    +-------------+
                    |  Match      |
                    |  Condition  |
                    +-------|-----+
                            |
            +-------------+-|-----------+-----------+
            |             |             |           |
            V             V             V           V
      ............  ............  ............ ...........
      : interface:  :    L2    :  :    L3    : :  L4     :  . . .
      :  match   :  :   match  :  :   match  : : match   :
      ''''''''''''  ''''''''''''  '''''''''''' '''''''''''
```

4. BNP Generic Info Model in High Level Yang

Below is the high level inclusion

```
      Figure 5
    module:pkt-eca-policy
     import ietf-inet-types  {prefix "inet"}
     import ietf-interface  {prefix "if"}
     import ietf-i2rs-rib   {prefix "iir"}

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

Below is the high level yang diagram

```
   module ietf-pkt-eca-policy
     +--rw pkt-eca-policy-cfg
     |  +--rw pkt-eca-policy-set
     |     +--rw groups* [group-name]
     |     |  +--rw group-name string
     |     |  +--rw vrf-name string
     |     |  +--rw address-family
     |     |  +--rw group-rule-list* [rule-name]
     |     |  |  +--rw rule-name
```

```
     |     |  |  +--rw rule-order-id
     |     |  |  +--rw default-action-id integer
     |     |  |  +--rw default-resolution-strategy-id integer
     |     +--rw rules* [order-id rule-name]
     |        +--rw order-id
     |        +--rw rule-name
     |        +--rw cfg-rule-conditions [cfgr-cnd-id]
     |        |  +--rw cfgr-cnd-id integer
     |        |  +--rw eca-event-match
     |        |  |  +--rw time-event-match*
     |        |  |  |   .. (time of day)
     |        |  +--rw eca-condition-match
     |        |  |  +--rw eca-pkt-matches*
     |        |  |  |  ... (L2-L4 matches)
     |        +--rw cfg-rule-actions [cfgr-action-id]
     |        |  +--rw cfgr-action-id
     |        |  +--rw eca-actions* [action-id]
     |        |  |  +--rw action-id uint32
     |        |  |  +--rw eca-ingress-act*
     |        |  |  | ... (permit, deny, mirror)
     |        |  |  +--rw eca-fwd-actions*
     |        |  |  | ...  (invoke, tunnel encap, fwd)
     |        |  |  +--rw eca-egress-act*
     |        |  |  | .. .
     |        |  |  +--rw eca-qos-actions*
     |        |  |  | ...
     |        |  |  +--rw ext-data-id integer
     |        +--rw cfg-external-data* [cfg-ext-data-id]
     |        |  +--rw cfg-ext-data-id integer
     |        |  +--rw data-type integer
     |        |  +--rw priority uint64
     |        |  |  uses external-data-forms
     |        |  ... (other external data)
     +--rw pkt-eca-policy-opstate
        +--rw pkt-eca-opstate
           +--rw groups* [group-name]
           |  +--rw rules-installed;
           |  +--rw rules_status* [rule-name]
                   |  +--rw strategy-used [strategy-id]
                   |  +--rw
           +--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-forward
                   +--rw op-external-data [op-ext-data-id]
                   |  +--rw op-ext-data-id integer
                   |  +--rw type identityref
                   |  +--rw installed-priority integer
                   |  |  (other details on external data )
```

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:   pkt-eca-policy-set:rules
external id:  pkt-eca-policy-set:cfg-external-data

Operational State for Policy
=======================================
Policy level: pkt-eca-policy-opstate
group level:  pkt-eca-opstate:groups
group-rule:   pkt-eca-opstate:rule-group-link*
rule level:   pkt-eca_opstate:rules_opstate*
              pkt-eca_op-stats
```

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: L2-header-match
     |     |  |  +--case: L3-header-match
     |     |  |  +--case: L4-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 L2-action
     |        |  |  +--case L3-action
     |        |  |  +--case L4-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@2017-03-13.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 "iir";
           }

   // 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 "2017-03-13" {
            description "third revision";
            reference "draft-ietf-i2rs-pkt-eca-policy-dm-03";
          }


   // interfaces - no identity  matches


         // 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 r
              type for GRE tunnel match ";
           }

           identity l3-icmp-header {
             base l3-header-match-type;
             description "L3 header match for ICMP";
            }

           identity l3-ipsec-ah-header {
             base l3-header-match-type;
             description "AH IPSEC header ";
            }

           identity l3-ipsec-esp-header {
             base l3-header-match-type;
             description "AH IPSEC header ";
            }

           // L4 header match identities

           identity l4-header-match-type {
            description "L4 header
            match types. (TCP, UDP,
            SCTP, UDPLite, 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-udplite {
             base l4-header-match-type;
             description "L4 header match for
               UDP lite";
            }

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

```
             }



           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 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 iir: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 iir: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-mpls-header {
                              uses iir:mpls-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-vsid {
           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;
                the 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 "ations 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 iir:ipv4-header;
                     }
                     case l3-ipv6-hdr {
                       uses iir:ipv6-header;
                     }
                     case L3-gre-tunnel {
                       uses iir: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 IPOv6 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 chunck
               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
                      stcp 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 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 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.";
                   }

        description "packet size by layer
            only non-zero values are matched";
           }

           grouping time-day-match {

         leaf hour {
               type uint8;
               description "hour
                   of day in 24  hours.
                   (add range)";
             }
             leaf minute {
                type uint8;
                    description
                    "minute in day.";
             }
```

```
             leaf second {
                type uint8;
                     description
                     "second in day.";
             }

           description "matches for
             time of day.";

           }



           grouping eca-event-matches {
              uses time-day-match;
              description "matches for events
               which include:
                   time of day.";

           }

       grouping eca-pkt-matches {
             uses interface-match;
             uses L2-header-match;
             uses L3-header-match;
             uses L4-header-match;
             uses packet-size-match;
             description "ECA matches";
           }

        grouping user-status-matches {
             leaf user {
              type string;
              description "user";
             }
             leaf region {
              type string;
              description "region";
              }
             leaf state {
              type string;
              description "state";
             }

             leaf user-status {
              type string;
              description "status of user";
             }
```

```
             description "user status
             matches - region,
             target, location";
           }

           grouping eca-condition-matches {
              uses eca-pkt-matches;
              uses user-status-matches;
              description "pkt
               and user status matches";
           }

           grouping eca-qos-actions {
             leaf cnt-actions {
               type uint32;
               description "count of ECA actions";
              }
             list qos-actions {
                key "action-id";
                leaf action-id {
                type uint32;
                description "action id";
                }
                uses interface-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,
                    L2, L3, and L4
                    headers.";
              }
             description "eca- qos actions";
            }

            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-ingress-actions {
             leaf permit {
              type boolean;
              description "permit ingress
               traffic.  False
                   means to deny.";
             }
             leaf mirror {
              type boolean;
              description "copy bytes
               ingressed to mirror port";
             }
             description "ingress eca match";
           }

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

     grouping eca-security-actions {
        leaf actions-exist {
           type boolean;
           description "existance of
            eca security actions";
        }
        description "content actions
             for security. Needs more
             description.";
     }

     grouping eca-egress-actions {
       leaf packet-rate {
             type uint32;
             description "maximum packet-rate";
       }
      leaf byte-rate {
```

```
             type uint64;
             description "maximum byte-rate ";
       }
       description "packet security actions";
     }


       grouping policy-conflict-resolution {
             list resolution-strategy {
              key "strategy-id";
              leaf strategy-id {
                type uint32;
                description "Id for strategy";
              }
              leaf stategy-name {
               type string;
                   description "name of strategy";
              }
              leaf filter-strategy {
                type string;
                description "type of resolution";

              }
              leaf global-strategy {
                type boolean;
                description "global strategy";
             }
             leaf mandatory-strategy {
              type boolean;
              description "required strategy";
             }
             leaf local-strategy {
              type boolean;
              description "local strategy";
             }
             leaf resolution-fcn {
              type uint64;
              description "resolution function id ";
             }
             leaf resolution-value {
              type uint64;
              description "resolution value";
             }
             leaf resolution-info {
              type string;
              description "resolution info";
             }
             list associate-ext-data {
```

```
                key "ext-data-id";
                    leaf ext-data-id {
                      type uint64;
                      description "ID of external data";
                    }
                    leaf ext-data {
                      type string;
                      description "external data";
                    }
                   description "linked external data";
             }
             description "list of strategies";
            }
             description "policy conflict
               resolution strategies";
           }


           grouping cfg-external-data {
             list cfg-ext-data {
               key "cfg-ext-data-id";
                leaf cfg-ext-data-id {
                type uint64;
                description "id for external data";
               }
               leaf data-type {
                type uint32;
                description "external data type ID";
                }
                leaf priority {
                 type uint64;
                 description "priority of data";
                }
               leaf other-data {
                type string;
                description "string
                     external data";
               }
             description "external data";
           }
            description "external data list";
       }


        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";
                   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-event-matches;
                     uses eca-ingress-actions;
                     uses eca-qos-actions;
                     uses eca-security-actions;
                     uses eca-fwd-actions;
                     uses eca-egress-actions;
                     uses cfg-external-data;
```

```
                     uses policy-conflict-resolution;

                     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 filter packets in a traffic stream, act to modify packets, and forward data packets. These filters operate dynamically at same level as currently deployed configured filter-based RIBs to filter, change, and forward traffic.

Due to the potential to use Filters as an attack vector, this data model should be used with the secure transport described in the [I-D.ietf-i2rs-protocol-security-requirements]

8. References

8.1. Normative References

[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-07 (work in progress), January 2017.

8.2. Informative References

[I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-17 (work in progress), September 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-10 (work in progress), December 2016.

[I-D.ietf-netmod-acl-model]
Bogdanovic, D., Koushik, K., Huang, L., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-10 (work in progress), March 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", draft-ietf-netmod-revised-datastores-00 (work in progress), December 2016.

[I-D.ietf-supa-generic-policy-data-model]
Halpern, J. and J. Strassner, "Generic Policy Data Model for Simplified Use of Policy Abstractions (SUPA)", draft-ietf-supa-generic-policy-data-model-02 (work in progress), October 2016.

[I-D.ietf-supa-generic-policy-info-model]
Strassner, J., Halpern, J., and S. Meer, "Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)", draft-ietf-supa-generic-policy-info-model-02 (work in progress), January 2017.

[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>.

[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, <http://www.rfc-editor.org/info/rfc7921>.

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

Authors’ Addresses

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Linda Dunbar
Huawei

Email: Linda.Dunbar@huawei.com

Russ White
Ericsson

Email: russw@riw.us

I2RS WG D. Migault
Internet-Draft J. Halpern
Intended status: Informational Ericsson
Expires: March 31, 2018 S. Hares
Huawei
September 27, 2017

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

Abstract

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

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

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

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

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

This document is structured as follows:

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

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

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

Motivations are described before the requirements are given.

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

2. Terminology and Acronyms

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

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

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

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

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

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

2.1. Requirements notation

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

3. I2RS Plane Isolation

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

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

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

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

3.1. I2RS Plane and Management plane

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

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

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

Security focus:

Data can be read by I2RS plane from configuration as copy of configuration data, or by management plane as copies of the I2RS plane. The problem is when the I2RS plane installs the routing plane as its new configuration or the management plane installs the I2RS plane information as management plane configuration. In this circumstance, we define "infecting" as interfering with and leading into a incoherent state. Planned interactions such as interactions denoted in in two cooperating Yang data modules is not an incoherent state.

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

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

APIs that interact with the
I2RS Plane and Management Plane

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

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

3.2. I2RS Plane and Forwarding Plane

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

```
    +--------------------------------------+
    |       Application                    |
    +--------------------------------------+
      || NB API                 NB API ||
      || I2RS client        mgt client ||
      ||                               ||
       I2RS protocol        mgt protocol
                            (NETCONF/RESTCONF)
```

figure 2

Applications may also communicate with multiple I2RS clients in order to have a broader view of the current and potential states of the network and the I2RS plane itself. These varied remote communication relationships between applications using the I2RS protocol to change the forwarding plane make it possible for an individual application to be an effective attack vector against the operation of the network, a router’s I2RS plane, the forwarding plane of the routing system, and other planes (management and control planes).

Prevention measures:

Systems should consider the following prevention errors:

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

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

3.3. I2RS Plane and Control Plane

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

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

Prevention measures:

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

3.4. Requirements

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

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

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

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

Explanation:

When the I2RS agent performs an action on a routing element, the action is performed in a process (or processes) associated with a routing process. For example, in a typical UNIX system, the user is designated with a user id (uid) and belongs to groups designated by group ids (gid). If such a user id (uid) and group id (gid) is the identifier for the routing processes peforming routing tasks in the control plane, then the I2RS Agent process would communicate with these routing processes. It is important that the I2RS agent has its own unique identifier and the routing processes have their own identifier so that access control can uniquely filter data between I2RS Agent and routing processes.

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

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

Explanation:

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

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

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

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

Explanation:

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

4. I2RS Access Control for Routing System Resources

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

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

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

4.1. I2RS Access Control Architecture

Overview:

Applications access the routing system resource via numerous intermediate nodes. The application communicates with an I2RS client. In some cases, the I2RS client is only associated with a single application attached to one or more agents (case a and case b in figure 4 below). In other cases, the I2RS client may be connected to two applications (case c in figure 4 below), or the I2RS may act as a broker (agent/client device shown in case d in figure 4 below). The I2RS client broker approach provides scalability to the I2RS architecture as it avoids each application being registered to the I2RS agent. Similarly, the I2RS access control should be able to scale to numerous applications.

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

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

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

b) I2RS application client pair talking to
two i2RS agents

```
                                  +--------+
+-------------+   +---------+     | I2RS   |
|   I2RS      |===| I2RS    |=====| agent 1|
|application 1|   | client 1|     +--------+
|             |   |         |     +--------+
|             |   |         |=====| I2RS   |
+-------------+   +---------+     | agent 2|
                                  +--------+
```

c) two applications talk to 1 client

```
                                 +--------+
+-------------+   +--------+     | I2RS   |
|   I2RS      |===|I2RS    |=====| agent 1|
|application 1|   |client 1|     +--------+
+-------------+   |        |     +--------+
+-------------+   |        |=====| I2RS   |
|  I2RS       |   |        |     | agent 2|
|application 2|===|        |     +--------+
+-------------+   +--------+
```

d) I2RS Broker (agent/client

```
                                +--------+
+-------------+  +--------+     | I2RS   |
|   I2RS      |==|I2RS    |=====|agent 3/|
|application 1|  |client 1|   ==|client 3+----+
+-------------+  +--------+   | +--+-----+    |
                              |    |          |
+-------------+  +--------+   |  +-------+ +--|----+
|  I2RS       |  |I2RS    |===|  |I2RS   | |I2RS   |
|application 2|==|client 2|      |agent 1| |agent 2|
+-------------+  +--------+      +-------+ +-------+
```

figure 3

4.1.1. Access control Enforcement Scope

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

Explanation:

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

4.1.2. Notification Requirements

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

Explanation:

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

SEC-REQ-07 indicates the I2RS agent may reject the request because the I2RS client is not an authorized I2RS client or lacks the privileges to perform the requested transaction (read, write, start notifications or logging). The I2RS client should be notified of the reason the I2RS agent rejected the transaction due to a lack of authorization or privileges, and the I2RS client should return a message to the application indicating the I2RS agent reject the transaction with an indication of this reason. Similarly, if the I2RS client does not grant the access to the application, the I2RS client should also inform the application. An example of an error message could be, "Read failure: you do not have read permission", "Write failure: you do not have write permission", or "Write failure: resource accessed by someone else".

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

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

4.1.3. Trust

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

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

Explanation:

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

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

4.1.4. Sharing access control Information

For the I2RS client:

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

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

Similarly, for the applications:

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

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

For both the application and the client:

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

Explanation:

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

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

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

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

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

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

Overview:

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

Requirements:

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

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

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

Explanation:

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

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

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

4.1.6. Managing Access Control Policy

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

Requirements:

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

Explanation:

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

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

Explanation:

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

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

Explanation:

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

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

Explanation:

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

4.2. I2RS Agent Access Control Policies

Overview:

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

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

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

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

4.2.1. I2RS Agent Access Control

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

Requirements:

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

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

SEC-ENV-REQ 24: The I2RS agent should know which identity (E.g. system user) performed the latest update of the routing resource. This is true for an identity inside and outside the I2RS plane, so the I2RS agent can appropriately perform an update according to the priorities associated to the requesting identity and the identity that last updated the resource.

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

Explanation:

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

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

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

4.2.2. I2RS Client Access Control Policies

Overview:

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

Requirements:

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

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

Explanation:

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

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

4.2.3. Application and Access Control Policies

Overview

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

Requirements:

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

Explanation:

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

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

Explanation:

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

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

Explanation:

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

5. I2RS Application Isolation

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

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

5.1. Robustness Toward Programmability

Overview

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

- the use of automation reduces configuration errors;

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

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

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

- o applications which belong to different tenants with different objectives,

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

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

Requirement:

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

Explanation:

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

5.2. Application Isolation

5.2.1. DoS

Overview:

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

Requirements:

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

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

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

Explanation:

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

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

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

5.2.2. Application Logic Control

Overview

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

Requirements:

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

Explanation:

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

6. Security Considerations

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

7. IANA Considerations

No IANA considerations for this requirements.

8. Acknowledgments

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

9. References

9.1. Normative References

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

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

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

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

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

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

9.2. Informative References

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

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

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

Authors' Addresses

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

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

Joel Halpern
Ericsson

Email: Joel.Halpern@ericsson.com

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 19, 2018

A. Clemm
Huawei USA
J. Medved
Cisco
R. Varga
Pantheon Technologies SRO
X. Liu
Jabil
H. Ananthakrishnan
Packet Design
N. Bahadur
Bracket Computing

December 16, 2017

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

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

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

2. Key Words

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

3. Definitions and Acronyms

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

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

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

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

LSP: Label Switched Path

NETCONF: Network Configuration Protocol

NMDA: Network Management Datastore Architecture

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

URI: Uniform Resource Identifier

SRLG: Shared Risk Link Group

TED: Traffic Engineering Database

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

4. Model Structure

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

```
                      +-----------------------------+
                      |  +-----------------------+  |
                      |  |      ietf-network     |  |
                      |  +----------^------------+  |
                      |             |               |
                      |  +-----------------------+  |
                      |  | ietf-network-topology |  |
                      |  +----------+------------+  |
                      +-------------^---------------+
                                    |
                                    |
                       +------------^-------------+
                       | ietf-l3-unicast-topology |
                       +------------^-------------+
                                    |
                                    |
                        +-----------^-----------+
                        | example-ospf-topology |
                        +-----------------------+
```

Figure 1: Overall model structure

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

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

5. Layer 3 Unicast Topology Model Overview

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

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

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

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

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

- 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.

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

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

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

6. Layer 3 Unicast Topology YANG Module

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

```
        WG List:   <mailto:i2rs@ietf.org>
        Editor:    Alexander Clemm
                   <mailto:ludwig@clemm.org>
        Editor:    Jan Medved
                   <mailto:jmedved@cisco.com>
        Editor:    Robert Varga
                   <mailto:robert.varga@pantheon.tech>
        Editor:    Xufeng Liu
                   <mailto:xliu@kuatrotech.com>
        Editor:    Nitin Bahadur
                   <mailto:nitin_bahadur@yahoo.com>
        Editor:    Hariharan Ananthakrishnan
                   <mailto:hari@packetdesign.com>";
     description
       "This module defines a model for Layer 3 Unicast
        topologies.
        Copyright (c) 2017 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.
        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust’s Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).
        This version of this YANG module is part of
        draft-ietf-i2rs-yang-l3-topology-16;
        see the RFC itself for full legal notices.
        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-l3-topology-16 with RFC
        number when published (i.e. RFC xxxx).";
     revision "2017-12-16" {
       description
         "Initial revision.
          NOTE TO RFC EDITOR: Please replace the following reference
          to draft-ietf-i2rs-yang-l3-topology-16 with
          RFC number when published (i.e. RFC xxxx).";
       reference
         "draft-ietf-i2rs-yang-l3-topology-16";
     }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<CODE ENDS>

7. Interactions with Other YANG Modules

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

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

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

8. IANA Considerations

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

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

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

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

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

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

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

9. Security Considerations

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

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

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

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

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

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

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

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

10. Contributors

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

- Vishnu Pavan Beeram, Juniper
- Igor Bryskin, Huawei
- Ken Gray, Cisco
- Aihua Guo, Huawei
- Tom Nadeau, Brocade
- Tony Tkacik
- Aleksandr Zhdankin, Cisco

11. Acknowledgements

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

12. References

12.1. Normative References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

12.2. Informative References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        Copyright (c) 2017 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.
        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust’s Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).
        This version of this YANG module is part of
        draft-ietf-i2rs-yang-l3-topology-16;
        see the RFC itself for full legal notices.
        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-l3-topology-16 with RFC
```

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

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

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

   <CODE ENDS>
```

Appendix B. Extending the Model

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

B.1. Example OSPF Topology

B.1.1. Model Overview

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

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

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

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

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

- Links are augmented with ospf link attributes.

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

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

B.1.2. OSPF Topology YANG Module

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

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

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

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

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

Appendix C. An Example

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

```
                +------------+                   +------------+
                |     D1     |                   |     D2     |
               /-\          /-\                 /-\          /-\
               | | 1-0-1    | |---------------->| | 2-1-1    | |
               | |    1-2-1 | |<----------------| |    2-0-1 | |
               \-/  1-3-1   \-/                 \-/  2-3-1   \-/
                |   /----\   |                   |   /----\   |
                +---|    |---+                   +---|    |---+
                    \----/                           \----/
                     A  |                             A  |
                     |  |                             |  |
                     |  |                             |  |
                     |  |       +------------+        |  |
                     |  |       |     D3     |        |  |
                     |  |      /-\          /-\       |  |
                     |  +----->| | 3-1-1    | |-------+  |
                     +---------| |    3-2-1 | |<---------+
                               \-/          \-/
                                |            |
                                +------------+
```

Figure 2: A network topology example

The corresponding instance data tree is depicted below:

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

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

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

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

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

Figure 3: Instance data tree

Authors' Addresses

Alexander Clemm
Huawei USA

EMail: ludwig@clemm.org

Jan Medved
Cisco

EMail: jmedved@cisco.com

Robert Varga
Pantheon Technologies SRO

EMail: robert.varga@pantheon.tech

Xufeng Liu
Jabil

EMail: Xufeng_Liu@jabil.com

Hariharan Ananthakrishnan
Packet Design

EMail: hari@packetdesign.com

Nitin Bahadur
Bracket Computing

EMail: nitin_bahadur@yahoo.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 21, 2018

A. Clemm
Huawei
J. Medved
Cisco
R. Varga
Pantheon Technologies SRO
N. Bahadur
Bracket Computing
H. Ananthakrishnan
Packet Design
X. Liu
Jabil

December 18, 2017

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

Abstract

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

Status of This Memo

This Internet-Draft will expire on June 21, 2018.

Copyright Notice

Table of Contents

1. Introduction

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

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

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

The basic data models introduced in this document are generic in nature and can be applied to many network and service topologies and inventories. The data models allow applications to operate on an inventory or topology of any network at a generic level, where the specifics of particular inventory/topology types are not required. At the same time, where data specific to a network type does comes into play and the data model is augmented, the instantiated data still adheres to the same structure and is represented in a consistent fashion. This also facilitates the representation of network hierarchies and dependencies between different network components and network types.

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

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

Figure 1: The network data model structure

The network-topology YANG module introduced in this document, entitled "ietf-network-topology.yang", defines a generic topology data model at its most general level of abstraction. The module defines a topology graph and components from which it is composed: nodes, edges and termination points. Nodes (from the ietf-network.yang module) represent graph vertices and links represent graph edges. Nodes also contain termination points that anchor the links. A network can contain multiple topologies, for example topologies at different layers and overlay topologies. The data model therefore allows to capture relationships between topologies, as well as dependencies between nodes and termination points across topologies. An example of a topology stack is shown in the following figure.

```
          +---------------------------------------+
         /            _[X1]_          "Service"  /
        /           _/  :   \_                  /
       /          _/     :    \_               /
      /         _/        :     \_            /
     /         /           :      \          /
    /       [X2]__________________[X3]      /
   +---------:--------------:------:-------+
              :              :     :
          +----:--------------:----:--------------+
         /      :              :   :        "L3" /
        /        :              :  :            /
       /         :               : :           /
      /         [Y1]_____________[Y2]         /
     /           *               * *         /
    /            *              *  *        /
   +--------------*-------------*--*-------+
                   *           *   *
          +--------*----------*----*--------------+
         /     [Z1]_______________[Z1] "Optical" /
        /         \_         *   _/             /
       /            \_      *  _/              /
      /               \_   * _/               /
     /                  \ * /                /
    /                    [Z]                /
   +---------------------------------------+
```

Figure 2: Topology hierarchy (stack) example

The figure shows three topology levels. At top, the "Service" topology shows relationships between service entities, such as service functions in a service chain. The "L3" topology shows network elements at Layer 3 (IP) and the "Optical" topology shows network elements at Layer 1. Service functions in the "Service" topology are mapped onto network elements in the "L3" topology, which in turn are mapped onto network elements in the "Optical" topology. The figure shows two Service Functions (X1 and X3) mapping onto a single L3 network element (Y2); this could happen, for example, if two service functions reside in the same VM (or server) and share the same set of network interfaces. The figure shows a single "L3" network element (Y2) mapped onto multiple "Optical" network elements (Z and Z1). This could happen, for example, if a single IP router attaches to multiple Reconfigurable Optical Add/Drop Multiplexers (ROADMs) in the optical domain.

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

```
                        VPN1                       VPN2
      +---------------------+    +---------------------+
     /   [Y5]...           /    / [Z5]______[Z3]      /
    /    /  \  :          /    /  : \_       / :     /
   /    /    \  :        /    /   :   \_    /  :    /
  /    /      \  :      /    /   :      \  /   :   /
 /   [Y4]____[Y1] :    /    /   :       [Z2]   :  /
+------:-------:---:--+    +---:---------:-----:-+
       :        :   :         :          :     :
       :         :   :       :           :     :
       :  +-------:---:-----:------------:-----:-----+
       : /       [X1]__:___:___________[X2]   :     /
       :/         / \_  : :       _____/ /   :     /
       :         /    \_ :  _____/      /   :     /
      /:        /       \: /           /   :     /
     / :       /        [X5]          /   :     /
    /   :     /       __/ \__        /   :     /
   /     :   /    ___/       \__    /   :     /
  /       : / ___/              \  /   :     /
 /        [X4]__________________[X3]..:     /
+------------------------------------------+
                               L3 Topology
```

Figure 3: Topology hierarchy (stack) example

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

There are multiple applications for such a data model. For example, within the context of I2RS, nodes within the network can use the data model to capture their understanding of the overall network topology and expose it to a network controller. A network controller can then use the instantiated topology data to compare and reconcile its own view of the network topology with that of the network elements that it controls. Alternatively, nodes within the network could propagate this understanding to compare and reconcile this understanding either among themselves or with help of a controller. Beyond the network element and the immediate context of I2RS itself, a network controller might even use the data model to represent its view of the topology that it controls and expose it to applications north of itself. Further use cases that the data model can be applied to are described in [I-D.draft-ietf-i2rs-usecase-reqs-summary].

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

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

2. Key Words

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

3. Definitions and Acronyms

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

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

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

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

URI: Uniform Resource Identifier

4. Model Structure Details

4.1. Base Network Model

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

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

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

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

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

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

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

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

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

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

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

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

4.2. Base Network Topology Data Model

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

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

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

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

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

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

4.3. Extending the data model

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

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

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

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

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

4.4. Discussion and selected design decisions

4.4.1. Container structure

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

4.4.2. Underlay hierarchies and mappings

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

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

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

4.4.3. Dealing with changes in underlay networks

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

A dangling leafref of a configured object leaves the corresponding instance in a state in which it lacks referential integrity, rendering it in effect inoperational. Any corresponding object instance is therefore removed from the operational state datastore until the situation has been resolved, i.e. until either the supporting object is added to the operational state datastore, or until the instance is reconfigured to refer to another object that is actually reflected in the operational state datastore. It does remain part of the intended datastore.

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

4.4.4. Use of groupings

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

4.4.5. Cardinality and directionality of links

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

4.4.6. Multihoming and link aggregation

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

4.4.7. Mapping redundancy

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

4.4.8. Typing

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

4.4.9. Representing the same device in multiple networks

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

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

```
                         +---------------------+
                        /   [X1]____[X2]      /  X(Service Overlay)
                       +----:--:----:--------+
                         ..:    :..: :
                ........:     ....: : :....
         +-----:-------------:--+    :     :...
        /   [Y1]____[Y2]....:  /      :..      :
       +------|-------|-------+          :..    :...
        Y(L3) |       +---------------------:-----+ :
              |                         +----:----|-:----------+
              +------------------------/---[D1]  [D2]         /
                                      +----------------------+
                                        P (Physical network)
```

Figure 6: Topology hierarchy example - multiple underlays

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

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

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

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

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

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

4.4.11. Identifiers of string or URI type

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

The case for strings is that they will be easier to implement. The reason for choosing URIs is that the topology/node/tp exists in a larger context, hence it is useful to be able to correlate identifiers across systems. While strings, being the universal data type, are easier for human beings, they also muddle things. What typically happens is that strings have some structure which is magically assigned and the knowledge of this structure has to be communicated to each system working with the data. A URI makes the structure explicit and also attaches additional semantics: the URI, unlike a free-form string, can be fed into a URI resolver, which can point to additional resources associated with the URI. This property is important when the topology data is integrated into a larger, more complex system.

5. Interactions with Other YANG Modules

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

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

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

6. YANG Modules

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        Copyright (c) 2017 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

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

        This version of this YANG module is part of
        draft-ietf-i2rs-yang-network-topo-20;
        see the RFC itself for full legal notices.

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

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

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

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

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

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

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

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

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

   <CODE ENDS>
```

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

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

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

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

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

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

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

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

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

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

      Editor:    Hariharan Ananthakrishnan
```

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

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

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

      Copyright (c) 2017 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

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

      This version of this YANG module is part of
      draft-ietf-i2rs-yang-network-topo-20;
      see the RFC itself for full legal notices.

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

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

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

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

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

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

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

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

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

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

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

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

   <CODE ENDS>
```

7. IANA Considerations

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

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

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

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

URI:urn:ietf:params:xml:ns:yang:ietf-network-topology-state
Registrant Contact: The IESG.

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

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

NOTE TO THE RFC EDITOR: In the list below, please replace references to "draft-ietf-i2rs-yang-network-topo-20 (RFC form)" with RFC number when published (i.e. RFC xxxx).

Name: ietf-network
Namespace: urn:ietf:params:xml:ns:yang:ietf-network
Prefix: nw
Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

Name: ietf-network-topology
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-topology
Prefix: nt
Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

Name: ietf-network-state
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-state
Prefix: nw-s
Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

Name: ietf-network-topology-state
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-topology-state
Prefix: nt-s
Reference: draft-ietf-i2rs-yang-network-topo-20.txt (RFC form)

8. Security Considerations

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

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

The network topology and inventory created by this module reveals information about the structure of networks that could be very helpful to an attacker. As a privacy consideration, while there is no personally identifiable information defined in this module, it is possible that some node identifiers may be associated with devices that are in turn associated with specific users.

The YANG modules define information that can be configurable in certain instances, for example in the case of overlay topologies that can be created by client applications. In such cases, a malicious client could introduce topologies that are undesired. Specifically, a malicious client could attempt to remove or add a node, a link, a termination point, by creating or deleting corresponding elements in the node, link, and termination point lists, respectively. In the case of a topology that is learned, the server will automatically prohibit such misconfiguration attempts. In the case of a topology that is configured, i.e. whose origin is "intended", the undesired configuration could become effective and be reflected in the operational state datastore, leading to disruption of services provided via this topology might be disrupted. For example, the topology could be "cut" or be configured in a suboptimal way, leading to increased consumption of resources in the underlay network due to resulting routing and bandwidth utilization inefficiencies. Likewise, it could lead to degradation of service levels as well as possibly disruption of service. For those reasons, it is important that the NETCONF access control model is vigorously applied to prevent topology misconfiguration by unauthorized clients.

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

- network: A malicious client could attempt to remove or add a network in an attempt to remove an overlay topology, or create an unauthorized overlay.

- supporting-network: A malicious client could attempt to disrupt the logical structure of the model, resulting in lack of overall data integrity and making it more difficult to, for example, troubleshoot problems rooted in the layering of network topologies.

- node: A malicious client could attempt to remove or add a node from network, for example in order to sabotage the topology of a network overlay.

- supporting-node: A malicious client could attempt to change the supporting-node in order to sabotage the layering of an overlay.

These are the subtrees and data nodes and their sensitivity/ vulnerability in the ietf-network-topology module:

- link: A malicious client could attempt to remove a link from a topology, or add a new link, or manipulate the way the link is layered over supporting links, or modify the source or destination of the link. Either way, the structure of the topology would be sabotaged, which could, for example, result in an overlay topology that is less than optimal.

- termination-point: A malicious client could attempt to remove termination points from a node, or add "phantom" termination points to a node, or change the layering dependencies of termination points, again in an attempt to sabotage the integrity of a topology and potentially disrupt orderly operations of an overlay.

9. Contributors

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

- Vishnu Pavan Beeram, Juniper

- Ken Gray, Cisco

- Tom Nadeau, Brocade

- Tony Tkacik

- Kent Watsen, Juniper

- Aleksandr Zhdankin, Cisco

10. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Alia Atlas, Andy Bierman, Martin Bjorklund, Igor Bryskin, Benoit Claise, Susan Hares, Ladislav Lhotka, Carlos Pignataro, Juergen Schoenwaelder, Robert Wilton, Qin Wu, and Xian Zhang.

11. References

11.1. Normative References

[I-D.draft-ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", I-D draft-ietf-netmod-revised-datastores-07, November 2017.

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

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

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

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

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

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

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

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

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

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

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

11.2. Informative References

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

[I-D.draft-ietf-i2rs-yang-l3-topology]
Clemm, A., Medved, J., Varga, R., Liu, X., Ananthakrishnan, H., and N. Bahadur, "A YANG Data Model for Layer 3 Topologies", I-D draft-ietf-i2rs-yang-l3-topology-16, December 2017.

[I-D.draft-ietf-netconf-yang-push]
Clemm, A., Voit, E., Gonzalez Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", I-D draft-ietf-netconf-yang-push-11, October 2017.

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

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

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

[RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, January 2003.

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

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, August 2016.

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

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

Appendix A. Model Use Cases

A.1. Fetching Topology from a Network Element

In its simplest form, topology is learned by a network element (e.g., a router) through its participation in peering protocols (IS-IS, BGP, etc.). This learned topology can then be exported (e.g., to a Network Management System) for external utilization. Typically, any network element in a domain can be queried for its topology and expected to return the same result.

In a slightly more complex form, the network element may be a controller, either by nature of it having satellite or subtended devices hanging off of it, or in the more classical sense, such as special device designated to orchestrate the activities of a number of other devices (e.g., an optical controller). In this case, the controller device is logically a singleton and must be queried distinctly.

It is worth noting that controllers can be built on top of controllers to establish a topology incorporating of all the domains within an entire network.

In all of the cases above, the topology learned by the network element is considered to be operational state data. That is, the data is accumulated purely by the network element’s interactions with other systems and is subject to change dynamically without input or consent.

A.2. Modifying TE Topology Imported from an Optical Controller

Consider a scenario where an Optical Transport Controller presents its topology in abstract TE Terms to a Client Packet Controller. This Customized Topology (that gets merged into the Client’s native topology) contains sufficient information for the path computing client to select paths across the optical domain according to its policies. If the Client determines (at any given point in time) that this imported topology does not exactly cater to its requirements, it may decide to request modifications to the topology. Such customization requests may include addition or deletion of topological elements or modification of attributes associated with existing topological elements. From the perspective of the Optical Controller, these requests translate into configuration changes to the exported abstract topology.

A.3. Annotating Topology for Local Computation

In certain scenarios, the topology learned by a controller needs to be augmented with additional attributes before running a computation algorithm on it. Consider the case where a path-computation application on the controller needs to take the geographic coordinates of the nodes into account while computing paths on the learned topology. If the learned topology does not contain these coordinates, then these additional attributes must be configured on the corresponding topological elements.

A.4. SDN Controller-Based Configuration of Overlays on Top of Underlays

In this scenario, an SDN controller (for example, Open Daylight) maintains a view of the topology of the network that it controls based on information that it discovers from the network. In addition, it provides an application in which it configures and maintains an overlay topology.

The SDN Controller thus maintains two roles:

- It is a client to the network.

- It is a server to its own northbound applications and clients, e.g. an OSS.

In other words, one system’s client (or controller, in this case) may be another system’s server (or managed system).

In this scenario, the SDN controller maintains a consolidated data model of multiple layers of topology. This includes the lower layers of the network topology, built from information that is discovered from the network. It also includes upper layers of topology overlay, configurable by the controller’s client, i.e. the OSS. To the OSS, the lower topology layers constitute "read-only" information. The upper topology layers need to be read-writable.

Appendix B. Companion YANG models for non-NMDA compliant implementations

The YANG modules defined in this document are designed to be used in conjunction with implementations that support the Network Management Datastore Architecture (NMDA) defined in [I-D.draft-ietf-netmod-revised-datastores]. In order to allow implementations to use the data model even in cases when NMDA is not supported, in the following two companion modules are defined that represent the operational state of networks and network topologies. The modules, ietf-network-state and ietf-network-topology-state, mirror modules ietf-network and ietf-network-topology defined earlier in this document. However, all data nodes are non-configurable. They represent state that comes into being by either learning topology information from the network, or by applying configuration from the mirrored modules.

The companion modules, ietf-network-state and ietf-network-topology-state, are redundant and SHOULD NOT be supported by implementations that support NMDA. It is for this reason that the definitions are defined in an appendix.

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

B.1. YANG Model for Network State

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

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

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

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

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

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

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

     Editor:    Robert Varga
```

```
                <mailto:robert.varga@pantheon.tech>

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

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

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

  description
    "This module defines a common base data model for a collection
     of nodes in a network. Node definitions are further used
     in network topologies and inventories.  It represents
     information that is either learned and automatically populated,
     or information that results from applying configured netwok
     information configured per the ietf-network data model,
     mirroring the corresponding data nodes in this data model.

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

     Copyright (c) 2017 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

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

     This version of this YANG module is part of
     draft-ietf-i2rs-yang-network-topo-20;
     see the RFC itself for full legal notices.

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

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

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

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

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

     container networks {
       config false;
       description
         "Serves as top-level container for a list of networks.";
       list network {
         key "network-id";
         description
           "Describes a network.
```

```
         A network typically contains an inventory of nodes,
         topological information (augmented through
         network-topology data model), as well as layering
         information.";
       container network-types {
         description
           "Serves as an augmentation target.
            The network type is indicated through corresponding
            presence containers augmented into this container.";
       }
       leaf network-id {
         type nw:network-id;
         description
           "Identifies a network.";
       }
       list supporting-network {
         key "network-ref";
         description
           "An underlay network, used to represent layered network
            topologies.";
         leaf network-ref {
           type leafref {
             path "/nw-s:networks/nw-s:network/nw-s:network-id";
           require-instance false;
           }
           description
             "References the underlay network.";
         }
       }
       list node {
         key "node-id";
         description
           "The inventory of nodes of this network.";
         leaf node-id {
           type nw:node-id;
           description
             "Identifies a node uniquely within the containing
              network.";
         }
         list supporting-node {
           key "network-ref node-ref";
           description
             "Represents another node, in an underlay network, that
              this node is supported by.  Used to represent layering
              structure.";
           leaf network-ref {
             type leafref {
               path "../../../nw-s:supporting-network/nw-s:network-ref";
```

```
            require-instance false;
            }
            description
              "References the underlay network that the
               underlay node is part of.";
          }
          leaf node-ref {
            type leafref {
              path "/nw-s:networks/nw-s:network/nw-s:node/nw-s:node-id";
            require-instance false;
            }
            description
              "References the underlay node itself.";
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

B.2. YANG Data Model for Network Topology State

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

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

    import ietf-network-state {
      prefix nw-s;
      reference
        "draft-ietf-i2rs-yang-network-topo-20
        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-network-topo-20 with RFC
        number when published (i.e. RFC xxxx).";
    }
    import ietf-network-topology {
      prefix nt;
      reference
        "draft-ietf-i2rs-yang-network-topo-20
        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-i2rs-yang-network-topo-20 with RFC
        number when published (i.e. RFC xxxx).";
    }
```

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

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

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

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

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

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

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

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

    description
      "This module defines a common base data model for network
       topology state, representing topology that is either learned,
       or topology that results from applying topology that has been
       configured per the ietf-network-topology data model, mirroring
       the corresponding data nodes in this data model. It augments
       the base network state data model with links to connect nodes,
       as well as termination points to terminate links on nodes.

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

       Copyright (c) 2017 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

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

```
       This version of this YANG module is part of
       draft-ietf-i2rs-yang-network-topo-20;
       see the RFC itself for full legal notices.

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

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

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

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

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

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

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

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

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

  <CODE ENDS>
```

Appendix C. An Example

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

```
                +------------+                   +------------+
                |     D1     |                   |     D2     |
               /-\          /-\                 /-\          /-\
               | | 1-0-1    | |---------------->| | 2-1-1    | |
               | |    1-2-1 | |<----------------| |    2-0-1 | |
               \-/  1-3-1   \-/                 \-/  2-3-1   \-/
                |   /----\   |                   |   /----\   |
                +---|    |---+                   +---|    |---+
                    \----/                           \----/
                     A  |                             A  |
                     |  |                             |  |
                     |  |                             |  |
                     |  |       +------------+        |  |
                     |  |       |     D3     |        |  |
                     |  |      /-\          /-\       |  |
                     |  +----->| | 3-1-1    | |-------+  |
                     +---------| |    3-2-1 | |<---------+
                               \-/          \-/
                                |            |
                                +------------+
```

Figure 7: A network topology example

The corresponding instance data tree is depicted below:

```
   {
     "ietf-network:networks": {
       "network": [
         {
           "network-types": {
           },
           "network-id": "otn-hc",
           "node": [
             {
               "node-id": "D1",
               "termination-point": [
                 {
                   "tp-id": "1-0-1"
                 },
                 {
                   "tp-id": "1-2-1"
                 },
                 {
                   "tp-id": "1-3-1"
                 }
               ]
             },
```

```
             {
               "node-id": "D2",
               "termination-point": [
                 {
                   "tp-id": "2-0-1"
                 },
                 {
                   "tp-id": "2-1-1"
                 },
                 {
                   "tp-id": "2-3-1"
                 }
               ]
             },
             {
               "node-id": "D3",
               "termination-point": [
                 {
                   "tp-id": "3-1-1"
                 },
                 {
                   "tp-id": "3-2-1"
                 }
               ]
             }
           ],
           "ietf-network-topology:link": [
             {
               "link-id": "D1,1-2-1,D2,2-1-1",
               "destination": {
                 "source-node": "D1",
                 "source-tp": "1-2-1"
               }
               "destination": {
                 "dest-node": "D2",
                 "dest-tp": "2-1-1"
               }
             },
             {
               "link-id": "D2,2-1-1,D1,1-2-1",
               "destination": {
                 "source-node": "D2",
                 "source-tp": "2-1-1"
               }
               "destination": {
                 "dest-node": "D1",
                 "dest-tp": "1-2-1"
               }
```

```
             },
             {
               "link-id": "D1,1-3-1,D3,3-1-1",
               "destination": {
                 "source-node": "D1",
                 "source-tp": "1-3-1"
               }
               "destination": {
                 "dest-node": "D3",
                 "dest-tp": "3-1-1"
               }
             },
             {
               "link-id": "D3,3-1-1,D1,1-3-1",
               "destination": {
                 "source-node": "D3",
                 "source-tp": "3-1-1"
               }
               "destination": {
                 "dest-node": "D1",
                 "dest-tp": "1-3-1"
               }
             },
             {
               "link-id": "D2,2-3-1,D3,3-2-1",
               "destination": {
                 "source-node": "D2",
                 "source-tp": "2-3-1"
               }
               "destination": {
                 "dest-node": "D3",
                 "dest-tp": "3-2-1"
               }
             },
             {
               "link-id": "D3,3-2-1,D2,2-3-1",
               "destination": {
                 "source-node": "D3",
                 "source-tp": "3-2-1"
               }
               "destination": {
                 "dest-node": "D2",
                 "dest-tp": "2-3-1"
               }
             }
           ]
         }
       ]
```

```
     }
   }
```

Figure 8: Instance data tree

Authors' Addresses

Alexander Clemm
Huawei

EMail: ludwig@clemm.org

Jan Medved
Cisco

EMail: jmedved@cisco.com

Robert Varga
Pantheon Technologies SRO

EMail: robert.varga@pantheon.tech

Nitin Bahadur
Bracket Computing

EMail: nitin_bahadur@yahoo.com

Hariharan Ananthakrishnan
Packet Design

EMail: hari@packetdesign.com

Xufeng Liu
Jabil

EMail: Xufeng_Liu@jabil.com

Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: July 17, 2018

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems

January 13, 2018

Network Management Datastore Architecture
draft-ietf-netmod-revised-datastores-10

Abstract

Datastores are a fundamental concept binding the data models written in the YANG data modeling language to network management protocols such as NETCONF and RESTCONF. This document defines an architectural framework for datastores based on the experience gained with the initial simpler model, addressing requirements that were not well supported in the initial model. This document updates RFC 7950.

Status of This Memo

This Internet-Draft will expire on July 17, 2018.

Copyright Notice

Table of Contents

1. Introduction

This document provides an architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [RFC8040] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding network management data models to network management protocols. Agreement on a common architectural model of datastores ensures that data models can be written in a network management protocol agnostic way. This architectural framework identifies a set of conceptual datastores but it does not mandate that all network management protocols expose all these conceptual datastores. This architecture is agnostic with regard to the encoding used by network management protocols.

This document updates RFC 7950 by refining the definition of the accessible tree for some XPath context (see Section 6.1) and the invocation context of operations (see Section 6.2).

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. Objectives

Network management data objects can often take two different values, the value configured by the user or an application (configuration) and the value that the device is actually using (operational state). These two values may be different for a number of reasons, e.g., system internal interactions with hardware, interaction with protocols or other devices, or simply the time it takes to propagate a configuration change to the software and hardware components of a system. Furthermore, configuration and operational state data objects may have different lifetimes.

The original model of datastores required these data objects to be modeled twice in the YANG schema, as "config true" objects and as "config false" objects. The convention adopted by the interfaces data model ([RFC7223]) and the IP data model ([RFC7277]) was using two separate branches rooted at the root of the data tree, one branch for configuration data objects and one branch for operational state data objects.

The duplication of definitions and the ad-hoc separation of operational state data from configuration data leads to a number of problems. Having configuration and operational state data in separate branches in the data model is operationally complicated and impacts the readability of module definitions. Furthermore, the relationship between the branches is not machine readable and filter expressions operating on configuration and on related operational state are different.

With the revised architectural model of datastores defined in this document, the data objects are defined only once in the YANG schema but independent instantiations can appear in different datastores, e.g., one for a configured value and another for an operationally used value. This provides a more elegant and simpler solution to the problem.

The revised architectural model of datastores supports additional datastores for systems that support more advanced processing chains converting configuration to operational state. For example, some systems support configuration that is not currently used (so called inactive configuration) or they support configuration templates that are used to expand configuration data via a common template.

3. Terminology

This document defines the following terminology. Some of the terms are revised definitions of terms originally defined in [RFC6241] and [RFC7950] (see also section Section 4). The revised definitions are semantically equivalent with the definitions found in [RFC6241] and [RFC7950]. It is expected that the revised definitions provided in this section will replace the definitions in [RFC6241] and [RFC7950] when these documents are revised.

- datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof. A datastore maps to an instantiated YANG data tree.

- schema node: A node in the schema tree. The formal definition is in RFC 7950.

- datastore schema: The combined set of schema nodes for all modules supported by a particular datastore, taking into consideration any deviations and enabled features for that datastore.

- configuration: Data that is required to get a device from its initial default state into a desired operational state. This data is modeled in YANG using "config true" nodes. Configuration can originate from different sources.

- configuration datastore: A datastore holding configuration.

- running configuration datastore: A configuration datastore holding the current configuration of the device. It may include configuration that requires further transformations before it can be applied. This datastore is referred to as "<running>".

- candidate configuration datastore: A configuration datastore that can be manipulated without impacting the device's running configuration datastore and that can be committed to the running configuration datastore. This datastore is referred to as "<candidate>".

- startup configuration datastore: A configuration datastore holding the configuration loaded by the device into the running configuration datastore when it boots. This datastore is referred to as "<startup>".

- intended configuration: Configuration that is intended to be used by the device. It represents the configuration after all configuration transformations to <running> have been performed and is the configuration that the system attempts to apply.

- intended configuration datastore: A configuration datastore holding the complete intended configuration of the device. This datastore is referred to as "<intended>".

- configuration transformation: The addition, modification or removal of configuration between the <running> and <intended> datastores. Examples of configuration transformations include the removal of inactive configuration and the configuration produced through the expansion of templates.

- conventional configuration datastore: One of the following set of configuration datastores: <running>, <startup>, <candidate>, and <intended>. These datastores share a common datastore schema, and protocol operations allow copying data between these datastores. The term "conventional" is chosen as a generic umbrella term for these datastores.

- conventional configuration: Configuration that is stored in any of the conventional configuration datastores.

- dynamic configuration datastore: A configuration datastore holding configuration obtained dynamically during the operation of a device through interaction with other systems, rather than through one of the conventional configuration datastores.

- dynamic configuration: Configuration obtained via a dynamic configuration datastore.

- learned configuration: Configuration that has been learned via protocol interactions with other systems and that is neither conventional nor dynamic configuration.

- system configuration: Configuration that is supplied by the device itself.

- default configuration: Configuration that is not explicitly provided but for which a value defined in the data model is used.

- applied configuration: Configuration that is actively in use by a device. Applied configuration originates from conventional, dynamic, learned, system and default configuration.

- system state: The additional data on a system that is not configuration, such as read-only status information and collected statistics. System state is transient and modified by interactions with internal components or other systems. System state is modeled in YANG using "config false" nodes.

- operational state: The combination of applied configuration and system state.

- operational state datastore: A datastore holding the complete operational state of the device. This datastore is referred to as "<operational>".

- origin: A metadata annotation indicating the origin of a data item.

- remnant configuration: Configuration that remains part of the applied configuration for a period of time after it has been removed from the intended configuration or dynamic configuration. The time period may be minimal, or may last until all resources used by the newly-deleted configuration (e.g., network connections, memory allocations, file handles) have been deallocated.

The following additional terms are not datastore specific but commonly used and thus defined here as well:

- client: An entity that can access YANG-defined data on a server, over some network management protocol.

- server: An entity that provides access to YANG-defined data to a client, over some network management protocol.

- notification: A server-initiated message indicating that a certain event has been recognized by the server.

- remote procedure call: An operation that can be invoked by a client on a server.

4. Background

NETCONF [RFC6241] provides the following definitions:

- datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof.

- configuration datastore: The datastore holding the complete set of configuration that is required to get a device from its initial default state into a desired operational state.

YANG 1.1 [RFC7950] provides the following refinements when NETCONF is used with YANG (which is the usual case but note that NETCONF was defined before YANG existed):

- datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.

- configuration datastore: When modeled with YANG, a configuration datastore is realized as an instantiated data tree with configuration.

[RFC6244] defined operational state data as follows:

- Operational state data is a set of data that has been obtained by the system at runtime and influences the system’s behavior similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.

Section 4.3.3 of [RFC6244] discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of [RFC6244] then concludes that at the time of the writing, modeling state as distinct leafs and distinct branches is the recommended approach.

Implementation experience and requests from operators [I-D.ietf-netmod-opstate-reqs], [I-D.openconfig-netmod-opstate] indicate that the datastore model initially designed for NETCONF and refined by YANG needs to be extended. In particular, the notion of intended configuration and applied configuration has developed.

4.1. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:

```
     +-------------+                 +-----------+
     | <candidate> |                 | <startup> |
     |  (ct, rw)   |<---+       +--->| (ct, rw)  |
     +-------------+    |       |    +-----------+
            |           |       |           |
            |         +-----------+         |
            +-------->| <running> |<--------+
                      | (ct, rw)  |
                      +-----------+
                            |
                            v
                     operational state  <--- control plane
                         (cf, ro)

     ct = config true; cf = config false
     rw = read-write; ro = read-only
     boxes denote datastores
```

Figure 1

Note that this diagram simplifies the model: read-only (ro) and read-write (rw) is to be understood at a conceptual level. In NETCONF, for example, support for <candidate> and <startup> is optional and <running> does not have to be writable. Furthermore, <startup> can only be modified by copying <running> to <startup> in the standardized NETCONF datastore editing model. The RESTCONF protocol does not expose these differences and instead provides only a writable unified datastore, which hides whether edits are done through <candidate> or by directly modifying <running> or via some other implementation specific mechanism. RESTCONF also hides how configuration is made persistent. Note that implementations may also have additional datastores that can propagate changes to <running>. NETCONF explicitly mentions so called named datastores.

Some observations:

- Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.

- The NETCONF <get> operation returns the contents of <running> together with the operational state. It is therefore necessary that "config false" data is in a different branch than the "config true" data if the operational state can have a different lifetime compared to configuration or if configuration is not immediately or successfully applied.

- Several implementations have proprietary mechanisms that allow clients to store inactive data in <running>. Inactive data is conceptually removed before validation.

- Some implementations have proprietary mechanisms that allow clients to define configuration templates in <running>. These templates are expanded automatically by the system, and the resulting configuration is applied internally.

- Some operators have reported that it is essential for them to be able to retrieve the configuration that has actually been successfully applied, which may be a subset or a superset of the <running> configuration.

5. Architectural Model of Datastores

Below is a new conceptual model of datastores extending the original model in order to reflect the experience gained with the original model.

```
     +-------------+                 +-----------+
     | <candidate> |                 | <startup> |
     |  (ct, rw)   |<---+       +--->| (ct, rw)  |
     +-------------+    |       |    +-----------+
            |           |       |           |
            |         +-----------+         |
            +-------->| <running> |<--------+
                      | (ct, rw)  |
                      +-----------+
                            |
                            |        // configuration transformations,
                            |        // e.g., removal of nodes marked as
                            |        // "inactive", expansion of
                            |        // templates
                            v
                      +------------+
                      | <intended> | // subject to validation
                      | (ct, ro)   |
                      +------------+
                            |        // changes applied, subject to
                            |        // local factors, e.g., missing
                            |        // resources, delays
                            |
       dynamic              |   +-------- learned configuration
       configuration        |   +-------- system configuration
       datastores -----+    |   +-------- default configuration
                       |    |   |
                       v    v   v
                    +---------------+
                    | <operational> | <-- system state
                    | (ct + cf, ro) |
                    +---------------+

     ct = config true; cf = config false
     rw = read-write; ro = read-only
     boxes denote named datastores
```

Figure 2

5.1. Conventional Configuration Datastores

The conventional configuration datastores are a set of configuration datastores that share exactly the same datastore schema, allowing data to be copied between them. The term is meant as a generic umbrella description of these datastores. If a module does not contain any configuration data nodes and it is not needed to satisfy any imports, then it MAY be omitted from the datastore schema for the conventional configuration datastores. The set of datastores include:

- <running>
- <candidate>
- <startup>
- <intended>

Other conventional configuration datastores may be defined in future documents.

The flow of data between these datastores is depicted in Section 5.

The specific protocols may define explicit operations to copy between these datastores, e.g., NETCONF defines the <copy-config> operation.

5.1.1. The Startup Configuration Datastore (<startup>)

The startup configuration datastore (<startup>) is a configuration datastore holding the configuration loaded by the device when it boots. <startup> is only present on devices that separate the startup configuration from the running configuration datastore.

The startup configuration datastore may not be supported by all protocols or implementations.

On devices that support non-volatile storage, the contents of <startup> will typically persist across reboots via that storage. At boot time, the device loads the saved startup configuration into <running>. To save a new startup configuration, data is copied to <startup>, either via implicit or explicit protocol operations.

5.1.2. The Candidate Configuration Datastore (<candidate>)

The candidate configuration datastore (<candidate>) is a configuration datastore that can be manipulated without impacting the device’s current configuration and that can be committed to <running>.

The candidate configuration datastore may not be supported by all protocols or implementations.

<candidate> does not typically persist across reboots, even in the presence of non-volatile storage. If <candidate> is stored using

non-volatile storage, it is reset at boot time to the contents of <running>.

5.1.3. The Running Configuration Datastore (<running>)

The running configuration datastore (<running>) is a configuration datastore that holds the current configuration of the device. It MAY include configuration that requires further transformation before it can be applied, e.g., inactive configuration, or template-mechanism-oriented configuration that needs further expansion. However, <running> MUST always be a valid configuration data tree, as defined in Section 8.1 of [RFC7950].

<running> MUST be supported if the device can be configured via conventional configuration datastores.

If a device does not have a distinct <startup> and non-volatile storage is available, the device will typically use that non-volatile storage to allow <running> to persist across reboots.

5.1.4. The Intended Configuration Datastore (<intended>)

The intended configuration datastore (<intended>) is a read-only configuration datastore. It represents the configuration after all configuration transformations to <running> are performed (e.g., template expansion, removal of inactive configuration), and is the configuration that the system attempts to apply.

<intended> is tightly coupled to <running>. Whenever data is written to <running>, then <intended> MUST also be immediately updated by performing all necessary configuration transformations to the contents of <running> and then <intended> is validated.

<intended> MAY also be updated independently of <running> if the effect of a configuration transformation changes, but <intended> MUST always be a valid configuration data tree, as defined in Section 8.1 of [RFC7950].

For simple implementations, <running> and <intended> are identical.

The contents of <intended> are also related to the "config true" subset of <operational>, and hence a client can determine to what extent the intended configuration is currently in use by checking whether the contents of <intended> also appear in <operational>.

<intended> does not persist across reboots; its relationship with <running> makes that unnecessary.

Currently there are no standard mechanisms defined that affect <intended> so that it would have different content than <running>, but this architecture allows for such mechanisms to be defined.

One example of such a mechanism is support for marking nodes as inactive in <running>. Inactive nodes are not copied to <intended>. A second example is support for templates, which can perform transformations on the configuration from <running> to the configuration written to <intended>.

5.2. Dynamic Configuration Datastores

The model recognizes the need for dynamic configuration datastores that are, by definition, not part of the persistent configuration of a device. In some contexts, these have been termed ephemeral datastores since the information is ephemeral, i.e., lost upon reboot. The dynamic configuration datastores interact with the rest of the system through <operational>.

The datastore schema for a dynamic configuration datastore MAY differ from the datastore schema used for conventional configuration datastores. If a module does not contain any configuration data nodes and it is not needed to satisfy any imports, then it MAY be omitted from the datastore schema for the dynamic configuration datastore.

5.3. The Operational State Datastore (<operational>)

The operational state datastore (<operational>) is a read-only datastore that consists of all "config true" and "config false" nodes defined in the datastore’s schema. In the original NETCONF model the operational state only had "config false" nodes. The reason for incorporating "config true" nodes here is to be able to expose all operational settings without having to replicate definitions in the data models.

<operational> contains system state and all configuration actually used by the system. This includes all applied configuration from <intended>, learned configuration, system-provided configuration, and default values defined by any supported data models. In addition, <operational> also contains applied configuration from dynamic configuration datastores.

The datastore schema for <operational> MUST be a superset of the combined datastore schema used in all configuration datastores except that configuration data nodes supported in a configuration datastore MAY be omitted from <operational> if a server is not able to accurately report them.

Requests to retrieve nodes from <operational> always return the value in use if the node exists, regardless of any default value specified in the YANG module. If no value is returned for a given node, then this implies that the node is not used by the device.

The interpretation of what constitutes as being "in use" by the system is dependent on both the schema definition and the device implementation. Generally, functionality that is enabled and operational on the system would be considered as being "in use". Conversely, functionality that is neither enabled nor operational on the system is considered as not being "in use", and hence SHOULD be omitted from <operational>.

<operational> SHOULD conform to any constraints specified in the data model, but given the principal aim of returning "in use" values, it is possible that constraints MAY be violated under some circumstances, e.g., an abnormal value is "in use", the structure of a list is being modified, or due to remnant configuration (see Section 5.3.1). Note, that deviations SHOULD be used when it is known in advance that a device does not fully conform to the <operational> schema.

Only semantic constraints MAY be violated, these are the YANG "when", "must", "mandatory", "unique", "min-elements", and "max-elements" statements; and the uniqueness of key values.

Syntactic constraints MUST NOT be violated, including hierarchical organization, identifiers, and type-based constraints. If a node in <operational> does not meet the syntactic constraints then it MUST NOT be returned, and some other mechanism should be used to flag the error.

<operational> does not persist across reboots.

5.3.1. Remnant Configuration

Changes to configuration may take time to percolate through to <operational>. During this period, <operational> may contain nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device. Such remnant configuration from the previous configuration persists until the system has released resources used by the newly-deleted configuration (e.g., network connections, memory allocations, file handles).

Remnant configuration is a common example of where the semantic constraints defined in the data model cannot be relied upon for <operational>, since the system may have remnant configuration whose constraints were valid with the previous configuration and that are not valid with the current configuration. Since constraints on "config false" nodes may refer to "config true" nodes, remnant configuration may force the violation of those constraints.

5.3.2. Missing Resources

Configuration in <intended> can refer to resources that are not available or otherwise not physically present. In these situations, these parts of <intended> are not applied. The data appears in <intended> but does not appear in <operational>.

A typical example is an interface configuration that refers to an interface that is not currently present. In such a situation, the interface configuration remains in <intended> but the interface configuration will not appear in <operational>.

Note that configuration validity cannot depend on the current state of such resources, since that would imply that removing a resource might render the configuration invalid. This is unacceptable, especially given that rebooting such a device would cause it to restart with an invalid configuration. Instead we allow configuration for missing resources to exist in <running> and <intended>, but it will not appear in <operational>.

5.3.3. System-controlled Resources

Sometimes resources are controlled by the device and the corresponding system controlled data appears in (and disappears from) <operational> dynamically. If a system controlled resource has matching configuration in <intended> when it appears, the system will try to apply the configuration, which causes the configuration to appear in <operational> eventually (if application of the configuration was successful).

5.3.4. Origin Metadata Annotation

As configuration flows into <operational>, it is conceptually marked with a metadata annotation ([RFC7952]) that indicates its origin. The origin applies to all configuration nodes except non-presence containers. The "origin" metadata annotation is defined in Section 7. The values are YANG identities. The following identities are defined:

- origin: abstract base identity from which the other origin identities are derived.
- intended: represents configuration provided by <intended>.

- dynamic: represents configuration provided by a dynamic configuration datastore.

- system: represents configuration provided by the system itself. Examples of system configuration include applied configuration for an always existing loopback interface, or interface configuration that is auto-created due to the hardware currently present in the device.

- learned: represents configuration that has been learned via protocol interactions with other systems, including protocols such as link-layer negotiations, routing protocols, DHCP, etc.

- default: represents configuration using a default value specified in the data model, using either values in the "default" statement or any values described in the "description" statement. The default origin is only used when the configuration has not been provided by any other source.

- unknown: represents configuration for which the system cannot identify the origin.

These identities can be further refined, e.g., there could be separate identities for particular types or instances of dynamic configuration datastores derived from "dynamic".

For all configuration data nodes in <operational>, the device SHOULD report the origin that most accurately reflects the source of the configuration that is in use by the system.

In cases where it could be ambiguous as to which origin should be used, i.e. where the same data node value has originated from multiple sources, then the description statement in the YANG module SHOULD be used as guidance for choosing the appropriate origin. For example:

If for a particular configuration node, the associated YANG description statement indicates that a protocol negotiated value overrides any configured value, then the origin would be reported as "learned", even when a learned value is the same as the configured value.

Conversely, if for a particular configuration node, the associated YANG description statement indicates that a protocol negotiated value does not override an explicitly configured value, then the origin would be reported as "intended" even when a learned value is the same as the configured value.

In the case that a device cannot provide an accurate origin for a particular configuration data node then it SHOULD use the origin "unknown".

6. Implications on YANG

6.1. XPath Context

This section updates section 6.4.1 of RFC 7950.

If a server implements the architecture defined in this document, the accessible trees for some XPath contexts are refined as follows:

- If the XPath expression is defined in a substatement to a data node that represents system state, the accessible tree is all operational state in the server. The root node has all top-level data nodes in all modules as children.

- If the XPath expression is defined in a substatement to a "notification" statement, the accessible tree is the notification instance and all operational state in the server. If the notification is defined on the top level in a module, then the root node has the node representing the notification being defined and all top-level data nodes in all modules as children. Otherwise, the root node has all top-level data nodes in all modules as children.

- If the XPath expression is defined in a substatement to an "input" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation instance and all operational state in the server. The root node has top-level data nodes in all modules as children. Additionally, for an RPC, the root node also has the node representing the RPC operation being defined as a child. The node representing the operation being defined has the operation’s input parameters as children.

- If the XPath expression is defined in a substatement to an "output" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation instance and all operational state in the server. The root node has top-level data nodes in all modules as children. Additionally, for an RPC, the root node also has the node representing the RPC operation being defined as a child. The node representing the operation being defined has the operation’s output parameters as children.

6.2. Invocation of Actions and RPCs

This section updates section 7.15 of RFC 7950.

Actions are always invoked in the context of the operational state datastore. The node for which the action is invoked MUST exist in the operational state datastore.

Note that this document does not constrain the result of invoking an RPC or action in any way. For example, an RPC might be defined to modify the contents of some datastore.

7. YANG Modules

```
<CODE BEGINS> file "ietf-datastores@2018-01-11.yang"

module ietf-datastores {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-datastores";
  prefix ds;

  organization
    "IETF Network Modeling (NETMOD) Working Group";

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

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

     Author:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

     Author:   Juergen Schoenwaelder
               <mailto:j.schoenwaelder@jacobs-university.de>

     Author:   Phil Shafer
               <mailto:phil@juniper.net>

     Author:   Kent Watsen
               <mailto:kwatsen@juniper.net>

     Author:   Rob Wilton
               <rwilton@cisco.com>";

  description
    "This YANG module defines two sets of identities for datastores.
     The first identifies the datastores themselves, the second
     identifies datastore properties.
```

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

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

        This version of this YANG module is part of RFC XXXX
        (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
        for full legal notices.";

     revision 2018-01-11 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: Network Management Datastore Architecture";
     }

     /*
      * Identities
      */

     identity datastore {
       description
         "Abstract base identity for datastore identities.";
     }

     identity conventional {
       base datastore;
       description
         "Abstract base identity for conventional configuration
          datastores.";
     }

     identity running {
       base conventional;
       description
         "The running configuration datastore.";
     }

     identity candidate {
       base conventional;
       description
         "The candidate configuration datastore.";
     }
```

```
     identity startup {
       base conventional;
       description
         "The startup configuration datastore.";
     }

     identity intended {
       base conventional;
       description
         "The intended configuration datastore.";
     }

     identity dynamic {
       base datastore;
       description
         "Abstract base identity for dynamic configuration datastores.";
     }

     identity operational {
       base datastore;
       description
         "The operational state datastore.";
     }

     /*
      * Type definitions
      */

     typedef datastore-ref {
       type identityref {
         base datastore;
       }
       description
         "A datastore identity reference.";
     }

   }

   <CODE ENDS>

   <CODE BEGINS> file "ietf-origin@2018-01-11.yang"

   module ietf-origin {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-origin";
     prefix or;

     import ietf-yang-metadata {
```

```
       prefix md;
     }

     organization
       "IETF Network Modeling (NETMOD) Working Group";

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

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

        Author:   Martin Bjorklund
                  <mailto:mbj@tail-f.com>

        Author:   Juergen Schoenwaelder
                  <mailto:j.schoenwaelder@jacobs-university.de>

        Author:   Phil Shafer
                  <mailto:phil@juniper.net>

        Author:   Kent Watsen
                  <mailto:kwatsen@juniper.net>

        Author:   Rob Wilton
                  <rwilton@cisco.com>";

     description
       "This YANG module defines an 'origin' metadata annotation, and a
        set of identities for the origin value.

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

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

        This version of this YANG module is part of RFC XXXX
        (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
        for full legal notices.";

     revision 2018-01-11 {
       description
         "Initial revision.";
       reference
```

```
        "RFC XXXX: Network Management Datastore Architecture";
     }

     /*
      * Identities
      */

     identity origin {
       description
         "Abstract base identity for the origin annotation.";
     }

     identity intended {
       base origin;
       description
         "Denotes configuration from the intended configuration
          datastore";
     }

     identity dynamic {
       base origin;
       description
         "Denotes configuration from a dynamic configuration
          datastore.";
     }

     identity system {
       base origin;
       description
         "Denotes configuration originated by the system itself.

          Examples of system configuration include applied configuration
          for an always existing loopback interface, or interface
          configuration that is auto-created due to the hardware
          currently present in the device.";
     }

     identity learned {
       base origin;
       description
         "Denotes configuration learned from protocol interactions with
          other devices, instead of via either the intended
          configuration datastore or any dynamic configuration
          datastore.

          Examples of protocols that provide learned configuration
          include link-layer negotiations, routing protocols, and
          DHCP.";
```

```
     }

     identity default {
       base origin;
       description
         "Denotes configuration that does not have an configured or
          learned value, but has a default value in use.  Covers both
          values defined in a 'default' statement, and values defined
          via an explanation in a 'description' statement.";
     }

     identity unknown {
       base origin;
       description
         "Denotes configuration for which the system cannot identify the
          origin.";
     }

     /*
      * Type definitions
      */

     typedef origin-ref {
       type identityref {
         base origin;
       }
       description
         "An origin identity reference.";
     }

     /*
      * Metadata annotations
      */

     md:annotation origin {
       type origin-ref;
       description
         "The 'origin' annotation can be present on any configuration
          data node in the operational state datastore.  It specifies
          from where the node originated.  If not specified for a given
          configuration data node then the origin is the same as the
          origin of its parent node in the data tree.  The origin for
          any top level configuration data nodes must be specified.";
     }
   }

   <CODE ENDS>
```

8. IANA Considerations

8.1. Updates to the IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

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

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

8.2. Updates to the YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:

```
name:         ietf-datastores
namespace:    urn:ietf:params:xml:ns:yang:ietf-datastores
prefix:       ds
reference:    RFC XXXX

name:         ietf-origin
namespace:    urn:ietf:params:xml:ns:yang:ietf-origin
prefix:       or
reference:    RFC XXXX
```

9. Security Considerations

This document discusses an architectural model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

Although this document specifies several YANG modules, these modules only define identities and a metadata annotation, hence the "YANG module security guidelines" do not apply.

The origin metadata annotation exposes the origin of values in the applied configuration. Origin information may provide hints that certain control plane protocols are active on a device. Since origin information is tied to applied configuration values, it is only accessible to clients that have the permissions to read the applied configuration values. Security administrators should consider the sensitivity of origin information while defining access control rules.

10. Acknowledgments

This document grew out of many discussions that took place since 2010. Several Internet-Drafts ([I-D.bjorklund-netmod-operational], [I-D.wilton-netmod-opstate-yang], [I-D.ietf-netmod-opstate-reqs], [I-D.kwatsen-netmod-opstate], [I-D.openconfig-netmod-opstate]) and [RFC6244] touched on some of the problems of the original datastore model. The following people were authors to these Internet-Drafts or otherwise actively involved in the discussions that led to this document:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Andy Bierman, YumaWorks, <andy@yumaworks.com>
- Marcus Hines, Google, <hines@google.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Balazs Lengyel, Ericsson, <balazs.lengyel@ericsson.com>
- Acee Lindem, Cisco Systems, <acee@cisco.com>
- Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- Tom Petch, Engineering Networks Ltd, <ietfc@btconnect.com>
- Anees Shaikh, Google, <aashaikh@google.com>
- Rob Shakir, Google, <robjs@google.com>
- Jason Sterne, Nokia, <jason.sterne@nokia.co>

Juergen Schoenwaelder was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

11. References

11.1. Normative References

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

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

[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>.

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

[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>.

11.2. Informative References

[I-D.bjorklund-netmod-operational]
Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", draft-bjorklund-netmod-operational-00 (work in progress), October 2012.

[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.kwatsen-netmod-opstate]
Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-02 (work in progress), February 2016.

[I-D.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[I-D.wilton-netmod-opstate-yang]
Wilton, R., ""With-config-state" Capability for NETCONF/RESTCONF", draft-wilton-netmod-opstate-yang-02 (work in progress), December 2015.

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

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

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <https://www.rfc-editor.org/info/rfc6244>.

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

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

Appendix A. Guidelines for Defining Datastores

The definition of a new datastore in this architecture should be provided in a document (e.g., an RFC) purposed to the definition of the datastore. When it makes sense, more than one datastore may be defined in the same document (e.g., when the datastores are logically connected). Each datastore’s definition should address the points specified in the sections below.

A.1. Define which YANG modules can be used in the datastore

Not all YANG modules may be used in all datastores. Some datastores may constrain which data models can be used in them. If it is desirable that a subset of all modules can be targeted to the datastore, then the documentation defining the datastore must indicate this.

A.2. Define which subset of YANG-modeled data applies

By default, the data in a datastore is modeled by all YANG statements in the available YANG modules. However, it is possible to specify criteria that YANG statements must satisfy in order to be present in a datastore. For instance, maybe only "config true" nodes, or "config false" nodes that also have a specific YANG extension, are present in the datastore.

A.3. Define how data is actualized

The new datastore must specify how it interacts with other datastores.

For example, the diagram in Section 5 depicts dynamic configuration datastores feeding into <operational>. How this interaction occurs has to be defined by the particular dynamic configuration datastores. In some cases, it may occur implicitly, as soon as the data is put into the dynamic configuration datastore while, in other cases, an explicit action (e.g., an RPC) may be required to trigger the application of the datastore’s data.

A.4. Define which protocols can be used

By default, it is assumed that both the NETCONF and RESTCONF protocols can be used to interact with a datastore. However, it may be that only a specific protocol can be used (e.g., ForCES) or that a subset of all protocol operations or capabilities are available (e.g., no locking or no XPath-based filtering).

A.5. Define YANG identities for the datastore

The datastore must be defined with a YANG identity that uses the "ds:datastore" identity, or one of its derived identities, as its base. This identity is necessary so that the datastore can be referenced in protocol operations (e.g., <get-data>).

The datastore may also be defined with an identity that uses the "or:origin" identity or one its derived identities as its base. This identity is needed if the datastore interacts with <operational> so that data originating from the datastore can be identified as such via the "origin" metadata attribute defined in Section 7.

An example of these guidelines in use is provided in Appendix B.

Appendix B. Ephemeral Dynamic Configuration Datastore Example

The section defines documentation for an example dynamic configuration datastore using the guidelines provided in Appendix A. While this example is very terse, it is expected to be that a standalone RFC would be needed when fully expanded.

This example defines a dynamic configuration datastore called "ephemeral", which is loosely modeled after the work done in the I2RS working group.

Name	Value
Name	ephemeral
YANG modules	all (default)
YANG nodes	all "config true" data nodes
How applied	changes automatically propagated to <operational>
Protocols	NC/RC (default)
YANG Module	(see below)

The example "ephemeral" datastore properties

```
   module example-ds-ephemeral {
     yang-version 1.1;
     namespace "urn:example:ds-ephemeral";
     prefix eph;

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

     // datastore identity
     identity ds-ephemeral {
       base ds:dynamic;
       description
         "The ephemeral dynamic configuration datastore.";
     }

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

Appendix C. Example Data

The use of datastores is complex, and many of the subtle effects are more easily presented using examples. This section presents a series of example data models with some sample contents of the various datastores.

C.1. System Example

In this example, the following fictional module is used:

```
   module example-system {
     yang-version 1.1;
     namespace urn:example:system;
     prefix sys;

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

```
       container system {
         leaf hostname {
           type string;
         }

         list interface {
           key name;

           leaf name {
             type string;
           }

           container auto-negotiation {
             leaf enabled {
               type boolean;
               default true;
             }
             leaf speed {
               type uint32;
               units mbps;
               description
                 "The advertised speed, in mbps.";
             }
           }

           leaf speed {
             type uint32;
             units mbps;
             config false;
             description
               "The speed of the interface, in mbps.";
           }

           list address {
             key ip;

             leaf ip {
               type inet:ip-address;
             }
             leaf prefix-length {
               type uint8;
             }
           }
         }
       }
     }
```

The operator has configured the host name and two interfaces, so the contents of <intended> are:

```
<system xmlns="urn:example:system">

  <hostname>foo.example.com</hostname>

  <interface>
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>64</prefix-length>
    </address>
  </interface>

  <interface>
    <name>eth1</name>
    <address>
      <ip>2001:db8::20</ip>
      <prefix-length>64</prefix-length>
    </address>
  </interface>

</system>
```

The system has detected that the hardware for one of the configured interfaces ("eth1") is not yet present, so the configuration for that interface is not applied. Further, the system has received a host name and an additional IP address for "eth0" over DHCP. In addition to a default value, a loopback interface is automatically added by the system, and the result of the "speed" auto-negotiation. All of this is reflected in <operational>. Note how the origin metadata attribute for several "config true" data nodes is inherited from their parent data nodes.

```
   <system
       xmlns="urn:example:system"
       xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

     <hostname or:origin="or:learned">bar.example.com</hostname>

     <interface or:origin="or:intended">
       <name>eth0</name>
       <auto-negotiation>
         <enabled or:origin="or:default">true</enabled>
         <speed>1000</speed>
       </auto-negotiation>
       <speed>100</speed>
       <address>
         <ip>2001:db8::10</ip>
         <prefix-length>64</prefix-length>
       </address>
       <address or:origin="or:learned">
         <ip>2001:db8::1:100</ip>
         <prefix-length>64</prefix-length>
       </address>
     </interface>

     <interface or:origin="or:system">
       <name>lo0</name>
       <address>
         <ip>::1</ip>
         <prefix-length>128</prefix-length>
       </address>
     </interface>

   </system>
```

C.2. BGP Example

Consider the following fragment of a fictional BGP module:

```
       container bgp {
         leaf local-as {
           type uint32;
         }
         leaf peer-as {
           type uint32;
         }
         list peer {
           key name;
           leaf name {
             type inet:ip-address;
           }
           leaf local-as {
             type uint32;
             description
               ".... Defaults to ../local-as";
           }
           leaf peer-as {
             type uint32;
             description
                "... Defaults to ../peer-as";
           }
           leaf local-port {
             type inet:port;
           }
           leaf remote-port {
             type inet:port;
             default 179;
           }
           leaf state {
             config false;
             type enumeration {
               enum init;
               enum established;
               enum closing;
             }
           }
         }
       }
```

In this example model, both bgp/peer/local-as and bgp/peer/peer-as have complex hierarchical values, allowing the user to specify default values for all peers in a single location.

The model also follows the pattern of fully integrating state ("config false") nodes with configuration ("config true") nodes. There is no separate "bgp-state" hierarchy, with the accompanying repetition of containment and naming nodes. This makes the model simpler and more readable.

C.2.1. Datastores

Each datastore represents differing views of these nodes. <running> will hold the configuration provided by the operator, for example a single BGP peer. <intended> will conceptually hold the data as validated, after the removal of data not intended for validation and after any local template mechanisms are performed. <operational> will show data from <intended> as well as any "config false" nodes.

C.2.2. Adding a Peer

If the user configures a single BGP peer, then that peer will be visible in both <running> and <intended>. It may also appear in <candidate>, if the server supports the candidate configuration datastore. Retrieving the peer will return only the user-specified values.

No time delay should exist between the appearance of the peer in <running> and <intended>.

In this scenario, we’ve added the following to <running>:

```
<bgp>
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
  </peer>
</bgp>
```

C.2.2.1. <operational>

The operational datastore will contain the fully expanded peer data, including "config false" nodes. In our example, this means the "state" node will appear.

In addition, <operational> will contain the "currently in use" values for all nodes. This means that local-as and peer-as will be populated even if they are not given values in <intended>. The value of bgp/local-as will be used if bgp/peer/local-as is not provided; bgp/peer-as and bgp/peer/peer-as will have the same relationship. In the operational view, this means that every peer will have values for their local-as and peer-as, even if those values are not explicitly configured but are provided by bgp/local-as and bgp/peer-as.

Each BGP peer has a TCP connection associated with it, using the values of local-port and remote-port from <intended>. If those values are not supplied, the system will select values. When the connection is established, <operational> will contain the current values for the local-port and remote-port nodes regardless of the origin. If the system has chosen the values, the "origin" attribute will be set to "system". Before the connection is established, one or both of the nodes may not appear, since the system may not yet have their values.

```
<bgp or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>established</state>
  </peer>
</bgp>
```

C.2.3. Removing a Peer

Changes to configuration may take time to percolate through the various software components involved. During this period, it is imperative to continue to give an accurate view of the working of the device. <operational> will contain nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device.

Consider the scenario where a client removes a BGP peer. When a peer is removed, the operational state will continue to reflect the existence of that peer until the peer’s resources are released, including closing the peer’s connection. During this period, the current data values will continue to be visible in <operational>, with the "origin" attribute set to indicate the origin of the original data.

```
<bgp or:origin="or:intended">
  <local-as>64501</local-as>
  <peer-as>64502</peer-as>
  <peer>
    <name>2001:db8::2:3</name>
    <local-as or:origin="or:default">64501</local-as>
    <peer-as or:origin="or:default">64502</peer-as>
    <local-port or:origin="or:system">60794</local-port>
    <remote-port or:origin="or:default">179</remote-port>
    <state>closing</state>
  </peer>
</bgp>
```

Once resources are released and the connection is closed, the peer’s data is removed from <operational>.

C.3. Interface Example

In this section, we will use this simple interface data model:

```
container interfaces {
  list interface {
    key name;
    leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf mtu {
      type uint16;
    }
    leaf-list ip-address {
      type inet:ip-address;
    }
  }
}
```

C.3.1. Pre-provisioned Interfaces

One common issue in networking devices is the support of Field Replaceable Units (FRUs) that can be inserted and removed from the device without requiring a reboot or interfering with normal operation. These FRUs are typically interface cards, and the devices support pre-provisioning of these interfaces.

If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point, then <intended> might contain the following:

```
<interfaces>
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
  </interface>
</interfaces>
```

Since the interface does not exist, this data does not appear in <operational>.

When a FRU containing this interface is inserted, the system will detect it and process the associated configuration. <operational> will contain the data from <intended>, as well as nodes added by the system, such as the current value of the interface’s MTU.

```
<interfaces or:origin="or:intended">
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
    <mtu or:origin="or:system">1500</mtu>
  </interface>
</interfaces>
```

If the FRU is removed, the interface data is removed from <operational>.

C.3.2. System-provided Interface

Imagine if the system provides a loopback interface (named "lo0") with a default ip-address of "127.0.0.1" and a default ip-address of "::1". The system will only provide configuration for this interface if there is no data for it in <intended>.

When no configuration for "lo0" appears in <intended>, then <operational> will show the system-provided data:

```
<interfaces or:origin="or:intended">
  <interface or:origin="or:system">
    <name>lo0</name>
    <ip-address>127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

When configuration for "lo0" does appear in <intended>, then <operational> will show that data with the origin set to "intended". If the "ip-address" is not provided, then the system-provided value will appear as follows:

```
<interfaces or:origin="or:intended">
  <interface>
    <name>lo0</name>
    <description>loopback</description>
    <ip-address or:origin="or:system">127.0.0.1</ip-address>
    <ip-address>::1</ip-address>
  </interface>
</interfaces>
```

Authors’ Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

Email: phil@juniper.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Robert Wilton
Cisco Systems

Email: rwilton@cisco.com

i2rs
Internet-Draft
Intended status: Informational
Expires: May 2, 2018

M. Wang, Ed.
Huawei
R. Gu
China Mobile
Victor. Lopez
Telefonica
S. Hu
China Mobile

October 29, 2017

Information Model of Control-Plane and User-Plane separation BNG

draft-wcg-i2rs-cu-separation-infor-model-02

Abstract

To improve network resource utilization and reduce the operation expense, the Control-Plane and User-Plane separation conception is raised [I-D.gu-nfvrg-cloud-bng-architecture]. This document describes the information model for the interface between Control-Plane and User-Plane separation BNG. This information model may involve both control channel interface and configuration channel interface. The interface for control channel allows the Control-Plane to send several flow tables to the User-Plane, such as user's information table, user's interface table, and user's QoS table, etc. And it also allows the User-Plane to report the resources and statistics information to the Control-Plane. The interface for configuration channel is in charge of the version negotiation of protocols between the Control-Plane and User-Plane, the configuration for devices of Control-Plane and User-Plane, and the reports of User-Plane's capabilities, etc. The information model defined in this document enables defining a standardized data model. Such a data model can be used to define an interface to the CU separation BNG.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The rapid development of new services, such as 4K, IoT, etc, and increasing numbers of home broadband service users present some new challenges for BNGs such as:

> Low resource utilization: The traditional BNG acts as both a gateway for user access authentication and accounting and an IP network’s Layer 3 edge. The mutually affecting nature of the tightly coupled control plane and forwarding plane makes it difficult to achieve the maximum performance of either plane.
>
> Complex management and maintenance: Due to the large numbers of traditional BNGs, a network must have each device configured one at a time when deploying global service policies. As the network expands and new services are introduced, this deployment mode will cease to be feasible as it is unable to manage services effectively and rectify faults rapidly.
>
> Slow service provisioning: The coupling of control plane and forwarding plane, in addition to a distributed network control mechanism, means that any new technology has to rely heavily on the existing network devices.

To address these challenges, cloud-based BNG with CU separation conception is raised [I-D.gu-nfvrg-cloud-bng-architecture]. The main idea of Control-Plane and User-Plane separation method is to extract and centralize the user management functions of multiple BNG devices, forming an unified and centralized control plane (CP). And the traditional router’s Control Plane and Forwarding Plane are both preserved on BNG devices in the form of a user plane (UP).

This document describes an information model for the interface between Control-Plane and User-Plane separation BNG. This information model may involve both control channel interface and configuration channel interface. The interface for control channel allows the Control-Plane to send several flow tables to the User-Plane, such as user’s information table, user’s interface table, and user’s QoS table, etc. And it also allows User-Plane to report the resources and statistics information to the Control-Plane. The interface for configuration channel is in charge of the version negotiation of protocols between the Control-Plane and User-Plane, the configuration for the devices of Control-Plane and User-Plane, and the report of User-Plane’s capabilities, etc. The information model defined in this document enables defining a standardized data model. Such a data model can be used to define an interface to the CU separation BNG.

2. Concept and Terminology

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

2.1. Terminology

BNG: Broadband Network Gateway. A broadband remote access server (BRAS, B-RAS or BBRAS) routes traffic to and from broadband remote access devices such as digital subscriber line access multiplexers (DSLAM) on an Internet service provider’s (ISP) network. BRAS can also be referred to as a Broadband Network Gateway (BNG).

CP: Control Plane. CP is a user control management component which supports the management of UP’s resources such as the user entry and forwarding policy

UP: User Plane. UP is a network edge and user policy implementation component. The traditional router’s Control Plane and Forwarding Plane are both preserved on BNG devices in the form of a user plane.

3. Control Plane and User Plane separation BNG Information Model Overview

Briefly, a CU separation BNG is made up of a centralized CP and a set of UPs. The CP is a user control management component which supports to manage UP’s resources such as the user entry and forwarding policy, for example, the access bandwidth and priority management. And the UP is a network edge and user policy implementation component. It can support the forwarding plane functions on traditional BNG devices, such as traffic forwarding, QoS, and traffic statistics collection, and it can also support the control plane functions on traditional BNG devices, such as routing, multicast, etc.

The following figure describes the architecture of CU separation BNG

```
             +-----+    +-----+    +-----+   +-----+
             |EMS  |    |DHCP |    |AAA  |   |policy
             |     |    |server    |server   |server
             +----|+    +---|-+    +--|--+   +--|--+
                  |         |         |         |
                  |         |         |         |
                  |         |         |         |
                  |         |         |         |
             +----|---------|---------|---------|----+
             | +--|----+ +--|--+  +---|--+ +----|--+ |
             | |address| | sub |  |  AAA | |service| |
             | |mgt    | | Mgt |  |      | |control| |
             | +-------+ +-----+  +------+ +-------+ |
             |                                       | Control Plane
             |   +--------------------------------+  |
             |   |     User plane management      |  |
             |   |                                |  |
             |   +-------------|------------------+  |
             +-----------------|---------------------+
                               |
                               |
                               |
              |----------------|-----------------|
              |                |                 |
              |                |                 |
     +--------|-----+   +------|-------+  +------|------+
     | +---------+  |   | +---------+  |  |+-----|----+ |
     | | routing |  |   | | routing |  |  || routing  | |
     | | control |  |   | | control |  |  || control  | |
     | +---------+  |   | +---------+  |  |+----------+ |
     | +----------+ |   | +----------+ |  |+----------+ |  User Plane
     | |forwarding| |   | |forwarding| |  ||forwarding| |
     | |plane     | |   | |plane     | |  ||plane     | |
     | +----------+ |   | +----------+ |  |+----------+ |
     +--------------+   +--------------+  +-------------+
```

The CU separated BNG is shown in above figure. The BNG Control Plane could be virtualized and centralized, which provides significant benefits such as centralized session management, flexible address allocation, high scalability for subscriber management capacity, and cost-efficient redundancy, etc. The functional components inside the BNG Service Control Plane can be implemented as VNFs and hosted in a NFVI.

The User Plane Management module in the BNG control plane centrally manages the distributed BNG User Planes (e.g. load balancing), as well as the setup, deletion, maintenance of channels between Control

Planes and User Planes. Other modules in the BNG control plane, such as address management, AAA, and etc., are responsible for the connection with outside subsystems in order to fulfill the service. The routing control and forwarding Plane in the BNG User Plane (local) could be distributed across the infrastructure.

3.1. Service Data Model Usage

The idea of the information model is to propose a set of generic and abstract information models. The models are intended to be used in both Control Plane and User Planes. A typical scenario would be that this model can be used as a compendium for the interface between Control Plane and User Planes of CU separation BNG, that corresponding data model or TLVs can be defined to realize the communication between the Control Plane and User Planes.

```
                    -----------------
                ////                 \\\\
            ////                         \\\\
          //              Cloud              \\
         |                                     |
        |                                       |
       |                                         |
       |                                         |
        |          +-----------------+          |
         |         |  Control Plane  |         |
          \\       |                 |       //
            \\\\   +---------+-------+   ////
                \\\\         |         ////
                    ------------------
                             |
         +------------------+-----------+-----+
         |                  |           |     |
   User’s information    IP address    QoS:  .......
   May Including:         ......       CIR;   |
   User ID;                 |          PIR;   |
   User MAC;                |          CBS;   |
   Access method(PPPoE,     |          PBS;   |
   IPoE, etc) ......        |         ......  |
         |                  |           |     |
         +------------------V-----------+-----+
                            |
                       +----+
                       |                                    -------
                       |                                 ///       \\\
+------+       +-------v---------+       +--------+     |             |
| OTL  |       |    User Plane   |       |  Core  |    |    Internet   |
|      +-------+                 +-------+ Routing|-----|             |
+------+       +-----------------+       +--------+      \\\       ///
                                                            -------
```

CU Separation BNG

As shown in above figure, when users access to the BNG network, the control plane solicits these users’ information (such as user’s ID, user’s MAC, user’s access methods, for example via PPPoE/IPoE), associates them with available bandwidth which are reported by User planes, and based on the service’s requirement to generate a set of tables, which may include user’s information, user’s IP address, and QoS, etc. Then the control plane can transmit these tables to the User planes. User planes receive these tables, parses it, matches these rules, and then performs corresponding actions.

4. Information Model

This section specifies the information model in Routing Backus-Naur Form [I-D.gu-nfvrg-cloud-bng-architecture]. This grammar intends to help readers better understand the English text description in order to derive a data model. However it may not provide all the details provided by the English text. When there is a lack of clarity in grammar the English text will take precedence.

This section describes information model that represents the concept of the interface of CU separation BNG which is languages and protocols neutral.

The following figure describes the Overview of Information Model for CU separation BNG.

```
<cu-separation-bng-infor-model>::=<control-plane-information-model>
                                  <user-plane-information-model>

<control-plane-information-model>::=<user-related-infor-model>
                                    <interface-related-infor-model>
                                    <device-related-infor-model>

<user-related-infor-model>::= <user-basic-information>
                              [<ipv4-informatiom>]|[<ipv6-information>]
                              [<qos-information>]

<user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                              [<ACCESS_TYPE>][<SESSION_ID>]
                              [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                              <USER_INTERFACE>

<ipv4-informatiom>::=<USER_ID><USER_IPV4>
                     <MASK_LENGTH><GATEWAY>
                     <VRF>

<ipv6-information>::=<USER_ID>(<USER_IPV6>
                     <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                     <VRF>

<qos-information>::=<USER_ID>
                    (<CIR><PIR><CBS><PBS>)
                    [<QOS_PROFILE>]

<interface-related-infor-model>::=<interface-information>

<interface-information>::=<IFINDEX><BAS_ENABLE>
                          <service-type>
```

```
 <service-type>::=<PPP_Only><IPV4_TRIG>
                  <IPV6_TRIG><ND-TRIG>
                  <ARP_PROXY>

 <device-related-infor-model>::=<address-field-distribute>

 <address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                              <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                              <IF_INDEX><MASK_LENGTH>

 <user-plane-information-model>::=<port-resources-infor-model>
                                  <traffic-statistics>

 <port-resource-information>::=<IF_INDEX><IF_NAME>
                               <IF_TYPE><LINK_TYPE>
                               <MAC_ADDRESS><IF_PHY_STATE>
                               <MTU>
 <traffic-statistics-information>::=<USER_ID><STATISTICS_TYPE>
                                    <INGRESS_STATIISTICS_PACKETS>
                                    <INGRESS STATISTICS_BYTES>
                                    <EGRESS_STATISTICS_PACKETS>
                                    <EGRESS_STATISTICS_BYTES>
```

4.1. Information Model for Control-Plane

This section describes information model for the Control-Plane (CP). As mentioned in section 3, the Control Plane is a user control management component which manages the user’s information, User-Plane’s resources and forwarding policy, etc. The control plane can generate several tables which contain a set of rules based on the resources and specific requirements of user’s service. After that, the control plane sends the tables to User Planes, and User planes receive the tables, parse them, match the rules, and then perform corresponding actions.

The Routing Backus-Naur Form grammar below illustrates the Information model for Control-Plane:

```
 <control-plane-information-model>::=<user-related-infor-model>
                                     <interface-related-infor-model>
                                     <device-related-infor-model>

 <user-related-infor-model>::= <user-basic-information>
                               [<ipv4-informatiom>]|[<ipv6-information>]
                               [<qos-information>]

 <user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                               [<ACCESS_TYPE>][<SESSION_ID>]
                               [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                               <USER_INTERFACE>

 <ipv4-informatiom>::=<USER_ID><USER_IPV4>
                      <MASK_LENGTH><GATEWAY>
                      <VRF>

 <ipv6-information>::=<USER_ID>(<USER_IPV6>
                      <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                      <VRF>

 <qos-information>::=<USER_ID>
                     (<CIR><PIR><CBS><PBS>)
                     [<QOS_PROFILE>]

 <interface-related-infor-model>::=<interface-information>

 <interface-information>::=<IFINDEX><BAS_ENABLE>
                           <service-type>

 <service-type>::=<PPP_Only><IPV4_TRIG>
                  <IPV6_TRIG><ND-TRIG>
                  <ARP_PROXY>

 <device-related-infor-model>::=<address-field-distribute>

 <address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                              <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                              <IF_INDEX><MASK_LENGTH>
```

user-related-infor-model: present the attributes which can describe the user’s profile, such as user’s basic information, qos, and IP address, etc.

interface-related-infor-model: present the attributes which relate to some physical/virtual interface. This model can be used to indicate which kinds of service can be supported by interfaces.

device-related-infor-model: present the attributes which relate to specific device. For example the control plane can manage and distribute the users, which belong to same subnet, to some specific devices. And the user plane’s devices provide corresponding service for these users.

4.1.1. User-Related Information

The user related information are a bunch of attributes which may bind to specific users. For example, the control plane can use a unified ID to distinguish different users and distribute the IP address and QoS rules to a specific user. In this section, the user related information models are presented. The user related information models include the user information model, IPv4/IPv6 information model, QoS information model, etc.

The Routing Backus-Naur Form grammar below illustrates the user related information model:

```
<user-related-infor-model>::= <user-basic-information>
                              [<ipv4-informatiom>]|[<ipv6-information>]
                              [<qos-information>]

<user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                              [<ACCESS_TYPE>][<SESSION_ID>]
                              [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                              <USER_INTERFACE>

<ipv4-informatiom>::=<USER_ID><USER_IPV4>
                     <MASK_LENGTH><GATEWAY>
                     <VRF>

<ipv6-information>::=<USER_ID>(<USER_IPV6>
                     <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                     <VRF>

<qos-information>::=<USER_ID>
                    (<CIR><PIR><CBS><PBS>)
                    [<QOS_PROFILE>]
```

4.1.1.1. User Basic Information Model

The User Basic Information model contains a set of attributes to describe the basic information of a specific user, such as user’s mac address, access type (via PPPoE, IPoE, etc), inner vlan ID, outer vlan ID, etc.

The Routing Backus-Naur Form grammar below illustrates the user basic information model:

```
 <user-basic-information> :: = <USER_ID> <MAC_ADDRESS>
                               [<ACCESS_TYPE>][<SESSION_ID>]
                               [<INNER_VLAN-ID>][<OUTER_VLAN_ID>]
                               <USER_INTERFACE>
```

USER_ID: is the identifier of user. This parameter is a unique and mandatory, it can be used to distinguish different users.

MAC_ADDRESS: is the MAC address of the user.

ACCESS_TYPE: This attribute is an optional parameter. It can be used to indicate the protocol be used for user’s accessing, such as PPPoE, IPoE, etc.

SESSION_ID: This attribute is an optional parameter. It can be used as the identifier of PPPoE session.

INNER_VLAN-ID: The identifier of user’s inner VLAN.

OUTER_VLAN_ID: The identifier of user’s outer VLAN.

USER_INTERFACE: This attribute specifies the binding interface of a specific user. The ifIndex of the interface MAY be included. This is the 32-bit ifIndex assigned to the interface by the device as specified by the Interfaces Group MIB [RFC2863]. The ifIndex can be utilized within a management domain to map to an actual interface, but it is also valuable in public applications [RFC5837]. The ifIndex can be used as an opaque token to discern which interface of User-Plane is providing corresponding service for specific user.

4.1.1.2. IPv4 Information Model

The IPv4 information model presents the user’s IPv4 parameters. It is an optional constructs. The Routing Backus-Naur Form grammar below illustrates the user’s IPv4 information model:

```
<ipv4-informatiom>::=<USER_ID><USER_IPV4>
                     <MASK_LENGTH><GATEWAY>
                     <VRF>
```

USER_ID: is the identifier of user. This parameter is unique and mandatory. This attribute is used to distinguish different users. And it collaborates with other IPv4 parameters to present the user’s IPv4 information.

USER_IPV4: This attribute specifies the user’s IPv4 address, and it’s usually used in user plane discovery and ARP reply message.

MASK_LENGTH: This attribute specifies the user’s subnet masks lengths which can identify a range of IP addresses that are on the same network.

GATEWAY: This attribute specifies the user’s gateway, and it’s usually used in User Plane discovery and ARP reply message.

VRF: is the identifier of VRF instance.

4.1.1.3. IPv6 Information Model

The IPv6 information model presents the user’s IPv6 parameters. It is an optional constructs. The Routing Backus-Naur Form grammar below illustrates the user’s IPv6 information model:

```
<ipv6-information>::=<USER_ID>(<USER_IPV6>
                     <PREFIX_LEN>)|(<PD_ADDRESS><PD_PREFIX_LEN>)
                     <VRF>
```

USER_ID: is the identifier of user. This parameter is unique and mandatory. This attribute is used to distinguish different users. And it collaborates with other IPv6 parameters to present the user’s IPv4 information.

USER_IPV6: This attribute specifies the user’s IPv6 address, and it usually be used in neighbor discovery (ND discovery).

PREFIX_LEN: This attribute specifies the user’s subnet prefix lengths which can identify a range of IP addresses that are on the same network.

PD_ADDRESS: In IPv6 networking, DHCPv6 prefix delegation is used to assign a network address prefix and automate configuration and provisioning of the public routable addresses for the network. This attribute specifies the user’s DHCPv6 prefix delegation address, and it’s usually used in neighbor discovery (ND discovery).

PD_PREFIX_LEN: This attribute specifies the user’s DHCPv6 delegation prefix length, and it’s usually used in neighbor discovery (ND discovery).

VRF: is the identifier of VRF instance

4.1.1.4. QoS Information Model

In CU separation BNG, the Control-Plane (CP) generates the QoS table base on UP's bandwidth resources and specific QoS requirements of user's services. This table contains a set of QoS matching rules such as user's committed information rate, peak information rate, committed burst size, etc. And it is an optional constructs. The Routing Backus-Naur Form grammar below illustrates the user's qos information model:

```
<qos-information>::=<USER_ID>
                    (<CIR><PIR><CBS><PBS>)
                    [<QOS_PROFILE>]
```

USER_ID: is the identifier of user. This parameter is unique and mandatory. This attribute is used to distinguish different users. And it collaborates with other qos parameters to present the user's qos information.

CIR: In BNG network, the Committed Information Rate (CIR) is the bandwidth for a user guaranteed by an internet service provider to work under normal conditions. This attribute is used to indicate the user's committed information rate, and it usually collaborates with other qos attributes (such as PIR, CBS, PBS, etc) to present the user's QoS profile.

PIR: Peak Information Rate (PIR) is a burstable rate set on routers and/or switches that allows throughput overhead. This attribute is used to indicate the user's peak information rate, and it usually collaborate with other QoS attributes (such as CIR, CBS, PBS, etc) to present the user's QoS profile.

CBS: The Committed Burst Size (CBS) specifies the relative amount of reserved buffers for a specific ingress network's forwarding class queue or egress network's forwarding class queue. This attribute is used to indicate the user's committed burst size, and it usually collaborates with other qos attributes (such as CIR, PIR, PBS, etc) to present the user's QoS profile.

PBS: The Peak Burst Size (PBS) sepcifies the maximum size of the first token bucket. This attribute is used to indicate the user's peak burst size, and it usually collaborate with other qos attributes (such as CIR, PIR, CBS, etc) to present the user's QoS profile.

QOS_PROFILE: This attribute specifies the standard profile provided by the operator. It can be used as a QoS template which is defined as a list of classes of services and associated properties. The properties may include:

- o Rate-limit: used to rate-limit the class of service. The value is expressed as a percentage of the global service bandwidth.

- o latency: used to define the latency constraint of the class. The latency constraint can be expressed as the lowest possible latency or a latency boundary expressed in milliseconds.

- o jitter: used to define the jitter constraint of the class. The jitter constraint can be expressed as the lowest possible jitter or a jitter boundary expressed in microseconds.

- o bandwidth: used to define a guaranteed amount of bandwidth for the class of service. It is expressed as a percentage.

4.1.2. Interface Related Information

This model contains the necessary information for the interface. It is used to indicate which kind of service can be supported by this interface. The Routing Backus-Naur Form grammar below illustrates the interface related information model:

```
 <interface-related-infor-model>::=<interface-information>

 <interface-information>::=<IFINDEX><BAS_ENABLE>
                           <service-type>

 <service-type>::=<PPP_Only><IPV4_TRIG>
                  <IPV6_TRIG><ND-TRIG>
                  <ARP_PROXY>
```

4.1.2.1. Interface Information Model

The interface model mentioned here is a logical construct that identifies a specific process or a type of network service. In CU separation BNG network, the Control-Plane (CP) generates the Interface-Infor table based on the available resources, which are received from the User-Plane (UP), and the specific requirements of user’s services.

The Routing Backus-Naur Form grammar below illustrates the interface information model:

```
    <interface-information>::=<IFINDEX><BAS_ENABLE>
                              <service-type>

    <service-type>::=<PPP_Only><IPV4_TRIG>
                     <IPV6_TRIG><ND-TRIG>
                     <ARP_PROXY>
```

IFINDEX: The IfIndex is the 32-bit index assigned to the interface by the device as specified by the Interfaces Group MIB [RFC2863]. The ifIndex can be utilized within a management domain to map to an actual interface, but it is also valuable in public applications. The ifIndex can be used as an opaque token to discern which interface of User-Plane is providing corresponding service for specific user.

BAS_ENABLE: This is a flag, and if it is TRUE, the BRAS is enabled on this interface.

PPP_Only: This is a flag, and if it is TRUE, the interface only supports PPP user.

IPV4_TRIG: This is a flag, and if it is TRUE, the interface supports that the user can be triggered to connect the internet by using IPv4 message.

IPV6_TRIG: This is a flag, and if it is TRUE, the interface supports that the user can be triggered to connect the internet by using IPv6 message.

ND-TRIG: This is a flag, and if it is TRUE, the interface supports that the user can be triggered to connect the internet by using neighbor discovery message.

ARP_PROXY: This is a flag, and if it is TRUE, the ARP PROXY is enabled on this interface.

4.1.3. Device Related Information

The device related information model presents the attributes which related to specific device. For example the control plane can manage and distribute the users, who belong to same subnet, to some specific devices. And then the user plane’s devices can provide corresponding service for these users. The Routing Backus-Naur Form grammar below illustrates the device related information model:

```
<device-related-infor-model>::=<address-field-distribute>

<address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                             <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                             <IF_INDEX><MASK_LENGTH>
```

4.1.3.1. Address field distribute Table

In CU separation BNG information model, the Control-Plane (CP) generates and sends this Address field distribute table to UP. Based on this table, the user-plane's devices can be divided into several blocks, and each block is in charge of working for users with the same subnet. The Routing Backus-Naur Form grammar below illustrates the address field distribute information model:

```
<address-field-distribute>::=<ADDRESS_SEGMENT><ADDRESS_SEGMENT_MASK>
                             <ADDRESS_SEGMENT_VRF><NEXT_HOP>
                             <IF_INDEX><MASK_LENGTH>
```

4.2. Information Model for User Plane

This section describes information model for the interface of User Plane (UP). As mentioned in section 3, the UP is a network edge and user policy implementation component. It supports: Forwarding plane functions on traditional BNG devices, including traffic forwarding, QoS, and traffic statistics collection and Control plane functions on traditional BNG devices, including routing, multicast, and MPLS.

In CU separation BNG information model, the CP generates tables and provides the rules. The UP plays two roles:

1. It receives these tables, parses it, and matches these rules, then performs corresponding actions.

2. It also generates several tables to report the available resources (such as usable interfaces, etc) and statistical information to CP.

The Routing Backus-Naur Form grammar below illustrates the User Plane information model:

```
<user-plane-information-model>::=<port-resources-infor-model>
                                 <traffic-statistics>

port-resource-information>::=<IF_INDEX><IF_NAME>
                             <IF_TYPE><LINK_TYPE>
                             <MAC_ADDRESS><IF_PHY_STATE>
                             <MTU>
<traffic-statistics-information>::=<USER_ID><STATISTICS_TYPE>
                                   <INGRESS_STATIISTICS_PACKETS>
                                   <INGRESS STATISTICS_BYTES>
                                   <EGRESS_STATISTICS_PACKETS>
                                   <EGRESS_STATISTICS_BYTES>
```

4.2.1. Port Resources of UP

The User Plane can generate the network resource table, which contains a bunch of attributes to present the available network resources, for example the usable interfaces.

The Figure below illustrates the Port Resources Information Table of User-Plane:

```
<port-resource-information>::<IF_INDEX><IF_NAME>
                             <IF_TYPE><LINK_TYPE>
                             <MAC_ADDRESS><IF_PHY_STATE>
                             <MTU>
```

IFINDEX: IfIndex is the 32-bit index assigned to the interface by the device as specified by the Interfaces Group MIB [RFC2863]. The ifIndex can be utilized within a management domain to map to an actual interface, but it is also valuable in public applications. The ifIndex can be used as an opaque token to discern which interface of User-Plane is available.

IF_NAME: the textual name of the interface. The value of this object should be the name of the interface as assigned by the local device and should be suitable for use in commands entered at the device’s ‘console’. This might be a text name, such as ‘le0’ or a simple port number, such as ‘1’, depending on the interface naming syntax of the device. If several entries in the ifTable together represent a single interface as named by the device, then each will have the same value of ifName.

IF_TYPE: the type of interface, such as Ethernet, GE, Eth-Trunk, etc.

LINK_TYPE: This attribute specifies the type of link, such as point-to-point, broadcast, multipoint, point-to-multipoint, private and public (accessibility and ownership), etc.

MAC_ADDRESS: This attribute specifies the available interface’s MAC address.

IF_PHY_STATE: The current operational state of the interface. This is an enumeration type node:

1- Up: ready to pass packets;

2- Down

3- Testing: in some test mode;

4- Unknow: status cannot be determined for some reason;

5- Dormant;

6- Not present: some component is missing.

MTU: This attribute specifies the available interface’s MTU (Maximum Transmission Unit).

4.2.2. Traffic Statistics Infor

The user-plane also generates the traffic statistics table to report the current traffic statistics.

The Figure below illustrates the Traffic Statistics Infor model of User-Plane:

```
<traffic-statistics-information>::=<USER_ID><STATISTICS_TYPE>
                                   <INGRESS_STATIISTICS_PACKETS>
                                   <INGRESS STATISTICS_BYTES>
                                   <EGRESS_STATISTICS_PACKETS>
                                   <EGRESS_STATISTICS_BYTES>
```

USER_ID: is the identifier of user. This parameter is unique and mandatory. This attribute is used to distinguish different users. And it collaborates with other statistics parameters such as ingress packets, egress packets, etc, to report the user’s status profile.

STATISTICS_TYPE: This attributes specifies the traffic type such as IPv4, IPv6, etc.

INGRESS_STATIISTICS_PACKETS: This attribute specifies the Ingress Statistics Packets of specific user.

INGRESS STATISTICS_BYTES: This attribute specifies the Ingress Statistics Bytes of specific user.

EGRESS_STATISTICS_PACKETS: This attribute specifies the Egress Statistics Packets of specific user.

EGRESS_STATISTICS_BYTES: This attribute specifies the Egress Statistics Bytes of specific user.

5. Security Considerations

None.

6. IANA Considerations

None.

7. Normative References

[I-D.gu-nfvrg-cloud-bng-architecture]
Gu, R. and S. Hu, "Control and User Plane Separation Architecture of BNG", draft-gu-nfvrg-cloud-bng-architecture-01 (work in progress), July 2017.

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

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, DOI 10.17487/RFC2863, June 2000, <https://www.rfc-editor.org/info/rfc2863>.

[RFC5837] Atlas, A., Ed., Bonica, R., Ed., Pignataro, C., Ed., Shen, N., and JR. Rivers, "Extending ICMP for Interface and Next-Hop Identification", RFC 5837, DOI 10.17487/RFC5837, April 2010, <https://www.rfc-editor.org/info/rfc5837>.

Authors’ Addresses

Michael Wang (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: wangzitao@huawei.com

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

Email: gurong_cmcc@outlook.com

Victor Lopez
Telefonica
Sur 3 building, 3rd floor, Ronda de la Comunicacion s/n
Madrid 28050
Spain

Email: victor.lopezalvarez@telefonica.com

Sujun Hu
China Mobile
32 Xuanwumen West Ave, Xicheng District
Beijing, Beijing 100053
China

Email: shujun_hu@outlook.com

Network Working Group R. White, Ed.
Internet-Draft S. Zandi, Ed.
Intended status: Informational LinkedIn
Expires: December 17, 2018 June 15, 2018

IS-IS Support for Openfabric
draft-white-openfabric-06

Abstract

Spine and leaf topologies are widely used in hyperscale and cloud scale networks. In most of these networks, configuration is automated, but difficult, and topology information is extracted through broad based connections. Policy is often integrated into the control plane, as well, making configuration, management, and troubleshooting difficult. Openfabric is an adaptation of an existing, widely deployed link state protocol, Intermediate System to Intermediate System (IS-IS) that is designed to:

- Provide a full view of the topology from a single point in the network to simplify operations
- Minimize configuration of each Intermediate System (IS) (also called a router or switch) in the network
- Optimize the operation of IS-IS within a spine and leaf fabric to enable scaling

This document begins with an overview of openfabric, including a description of what may be removed from IS-IS to enable scaling. The document then describes an optimized adjacency formation process; an optimized flooding scheme; some thoughts on the operation of openfabric, metrics, and aggregation; and finally a description of the changes to the IS-IS protocol required for openfabric.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. Goals

Spine and leaf fabrics are often used in large scale data centers; in this application, they are commonly called a fabric because of their regular structure and predictable forwarding and convergence properties. This document describes modifications to the IS-IS protocol to enable it to run efficiently on a large scale spine and leaf fabric, openfabric. The goals of this control plane are:

- Provide a full view of the topology from a single point in the network to simplify operations
- Minimize configuration of each IS in the network
- Optimize the operation of IS-IS within a spine and leaf fabric to enable scaling

1.2. Contributors

The following people have contributed to this draft: Nikos Triantafillis (reflected flooding optimization), Ivan Pepelnjak (fabric locality calculation modifications), Christian Franke (fabric localigy calculation modification), Hannes Gredler (do not reflood optimizations), Les Ginsberg (capabilities encoding, circuit local reflooding), Naiming Shen (capabilities encoding, circuit local reflooding), Uma Chunduri (failure mode suggestions, flooding), Nick Russo, and Rodny Molina.

See [RFC5449], [RFC5614], and [RFC7182] for similar solutions in the Mobile Ad Hoc Networking (MANET) solution space.

1.3. Simplification

In building any scalable system, it is often best to begin by removing what is not needed. In this spirit, openfabric implementations MAY remove the following from IS-IS:

- External metrics. There is no need for external metrics in large scale spine and leaf fabrics; it is assumed that metrics will be properly configured by the operator to account for the correct order of route preference at any route redistribution point.
- Tags and traffic engineering processing. Openfabric is only designed to provide topology and reachability information. It is not designed to provide for traffic engineering, route preference through tags, or other policy mechanisms. It is assumed that all

routing policy will be provided through an overlay system which communicates directly with each IS in the fabric, such as PCEP [RFC5440] or I2RS [RFC7921]. Traffic engineering is assumed to be provided through Segment Routing (SR) [I-D.ietf-spring-segment-routing].

1.4. Additions and Requirements

To create a scalable link state fabric, openfabric includes the following:

- A slightly modified adjacency formation process.
- Mechanisms for determining which tier within a spine and leaf fabric in which the IS is located.
- A mechanism that reduces flooding to the minimum possible, while still ensuring complete database synchronization among the intermediate systems within the fabric.

Three general requirements are placed here; more specific requirements are considered in the following sections. Openfabric implementations:

- MUST support [RFC5301] and enable hostname advertisement by default if a hostname is configured on the intermediate system.
- SHOULD support [RFC6232], purge originator identification for IS-IS.
- MUST NOT be mixed with standard IS-IS implementations in operational deployments. Openfabric and standard IS-IS implementations SHOULD be treated as two separate protocols.

1.5. Sample Network

The following spine and leaf fabric will be used to describe these modifications.

```
+----+ +----+ +----+ +----+ +----+ +----+
| 1A | | 1B | | 1C | | 1D | | 1E | | 1F | (T0)
+----+ +----+ +----+ +----+ +----+ +----+

+----+ +----+ +----+ +----+ +----+ +----+
| 2A | | 2B | | 2C | | 2D | | 2E | | 2F | (T1)
+----+ +----+ +----+ +----+ +----+ +----+

+----+ +----+ +----+ +----+ +----+ +----+
| 3A | | 3B | | 3C | | 3D | | 3E | | 3F | (T2)
+----+ +----+ +----+ +----+ +----+ +----+

+----+ +----+ +----+ +----+ +----+ +----+
| 4A | | 4B | | 4C | | 4D | | 4E | | 4F | (T1)
+----+ +----+ +----+ +----+ +----+ +----+

+----+ +----+ +----+ +----+ +----+ +----+
| 5A | | 5B | | 5C | | 5D | | 5E | | 5F | (T0)
+----+ +----+ +----+ +----+ +----+ +----+
```

Figure 1

To reduce confusion (spine and leaf fabrics are difficult to draw in plain text art), this diagram does not contain the connections between devices. The reader should assume that each device in a given layer is connected to every device in the layer above it. For instance:

- 5A is connected to 4A, 4B, 4C, 4D, 4E, and 4F

- 5B is connected to 4A, 4B, 4C, 4D, 4E, and 4F

- 4A is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F

- 4B is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F

- etc.

The tiers or stages of the fabric are also marked for easier reference. T0 is assumed to be connected to application servers, or rather they are Top of Rack (ToR) intermediate systems. The remaining tiers, T1 and T2, are connected only to the fabric itself. Note there are no "cross links," or "east west" links in the illustrated fabric. The fabric locality detection mechanism described here will not work if there are cross links running east/

west through the fabric. Locality detection may be possible in such a fabric; this is an area for further study.

2. Modified Adjacency Formation

Because Openfabric operates in a tightly controlled data center environment, various modifications can be made to the IS-IS neighbor formation process to increase efficencicy and simplify the protocol. Specifically, Openfabric implementations SHOULD support [RFC3719], section 4, hello padding for IS-IS. Variable hello padding SHOULD NOT be used, as data center fabrics are built using high speed links on which padded hellos will have little performance impact. Further modifications to the neighbor formation process are considered in the following sections.

2.1. Level 2 Adjacencies Only

Openfabric is designed to work in a single flooding domain over a single data center fabric at the scale of thousands of routers with hundreds of thousands of routes (so a moderate scale in router and route count terms). Because of the way Openfabric optimizes operation in this environment, it is not necessary nor desirable to build multiple flooding domains. For instance, the flooding optimizations described later this document require a full view of the topology, as does any proposed overlay to inject policy into the forwarding plane. In light of this, the following changes SHOULD BE to IS-IS implemetations to support Openfabric:

- IIH PDU 17 (level 2 point-to-point circuit hello) should be the only IIH PDU type transmitted (see section 9.7 of ISO 10589)

- In IIH PDU 17 (level 2 point-to-point circuit hello), the Circuit Type field should be set to 2 (see section 9.7 of ISO 10589)

- Support for IIH PDU 15 (level 1 broadcast hello) should be removed (see section 9.5 of ISO 10589)

- Support for IIH PDU 16 (level 2 broadcast hello) should be removed (see section 9.6 of ISO 10589)

2.2. Point-to-point Adjacencies

Data center network fabrics only contain point-to-point links; because of this, there is no reason to support any broadcast link types, nor to support the Designated Intermediate System processing, including pseudonode creation. In light ot his, processing related to sections 7.2.3 (broadcast networks), 7.3.8 (generation of level 1 pseudonode LSPs), 7.3.10 (generation of level 2 pseudonode LSPs), and section 8.4.5 (LAN designated intermediate systems) in [ISO10589] SHOULD BE removed.

2.3. Three Way Handshake Support

It is important that two way connectivity be established before synchronizing the link state database, or routing through a link in a data center fabric. To reject optical failures that cause a one way connection between two routers, fabricDC must support the three way handshake mechanism described in [RFC5303].

2.4. Adjacency Formation Optimization

While adjacency formation is not considered particularly burdensome in IS-IS, it may still be useful to reduce the amount of state transferred across the network when connecting a new IS to the fabric. In its simplest form, the process is:

- An IS connected to the fabric will send hellos on all links.
- The IS will only complete the three-way handshake with one newly discovered neighbor; this would normally be the first neighbor which sends the newly connected intermediate system’s ID back in the three-way handshake process.
- The IS will complete its database exchange with this one newly adjacent neighbor.
- Once this process is completed, the IS will continue processing the remaining neighbors as normal.
- If synchronization is not achieved within twice the dead timer on the local interface, the newly connected IS will repeat this process with the second neighbor with which it forms a three-way adjacency.

This process allows each IS newly added to the fabric to exchange a full table once; a very minimal amount of information will be transferred with the remaining neighbors to reach full synchronization.

Any such optimization is bound to present a tradeoff between several factors; the mechanism described here increases the amount of time required to form adjacencies slightly in order to reduce the total state carried across the network. An alternative mechanism could provide a better balance of the amount of information carried across the network for initial synchronization and the time required to synchronize a new IS. For instance, an IS could choose to synchronize its database with two or three adjacent intermediate systems, which could speed the synchronization process up at the cost of carrying additional data on the network. A locally determined balance between the speed of synchronization and the amount of data carried on the network can be acheived by adjusting the number of adjacent intermediate systems the newly attached IS synchronizes with.

3. Advertisement of Reachability Information

IS-IS describes the topology in two different sets of TLVs; the first describes the set of neighbors connected to an IS, the second describes the set of reachable destination connected to an IS. There are two different forms of both of these descriptions, one of which carries what are widely called narrow metrics, the other of which carries what are widely called wide metrics. In a tightly controlled data center fabric implementation, such as the ones Openfabric is designed to support, no IS that supports narrow metrics will ever be deployed or supported; hence there is no reason to support any metric type other than wide metrics.

- The Level 2 Link State PDU (type 20 in section 9.9 of [ISO10589]) and the scoped flooding PDU (type 10 in section 3.1 of [RFC7356]) SHOULD BE the only PDU types used to carry link state information in a Openfabric implementation

- Processing related to the Level 1 Link State PDU (type 18) MAY BE removed from Openfabric implementations (see section 9.8 of [ISO10589])

- Neighbor reachability MUST BE carried in TLV type 22 (see section 3 of [RFC5305])

- IPv4 reachability SHOULD BE carried in TLV type 135 (see section 4 of [RFC5305]), or TLV type 235 for multitopology implementations (see [RFC5120])

- IPv6 reachability SHOULD BE carried in TLV type 236 (see [RFC5308]), or TLV type 237 for multitopology implemenations (see [RFC5120])

- Processing related to the neighbor reachability TLV (type 2, see sections 9.8 and 9.9 of [ISO10589]) SHOULD BE removed

- Processing related to the narrow metric IP reachability TLV (types 128 and 130) SHOULD BE removed

Further, if segment routing support is desired, Openfabric MAY support the Prefix Segment Identifier sub-TLV and other TLVs as required in [I-D.ietf-isis-segment-routing-extensions].

4. Determining and Advertising Location on the Fabric

The tier to which a IS is connected is useful to enable autoconfiguration of intermediate systems connected to the fabric and to reduce flooding. Once the tier of an intermediate system within the fabric has been determined, it MUST be advertised using the 4 bit Tier field described in section 3.3 of [I-D.shen-isis-spine-leaf-ext]. This section describes a method of calculating the tier number, assuming the tier numbers rise in value from the edge of the fabric.

This method begins with two of the T0 intermediate systems advertising their location in the fabric. This information can either be obtained through:

- Two T0 intermediate systems are manually configured to advertise 0x00 in their IS reachability tier sub-TLV, indicating they are at the edge of the fabric (a ToR IS).

- The T0 intermediate systems detect they are T0 through the presence connected hosts (i.e. through a request for address assignment or some other means). If such detection is used, and the IS determines it is located at T0, it should advertise 0x00 in its IS reachability tier sub-TLV.

If the first method is used, the two T0 routers MUST be "maximally separated" on the fabric. They must be a maximal number of hops apart, or rather thay MUST NOT be connected to the same T1 device as their "upstream" towards the superspines in a 5 ary fabric.

The second method above SHOULD be used with care, as it may not be secure, and it may not work in all data center environments. For instance, if a host is mistakenly (or intentionally, as a form of attack) attached to a spine IS, or a request for address assignment is transmitted to a spine IS during the bootup phase of the device or fabric, it is possible to cause a spine IS to advertise itself as a T0. Unless the autodetection of the T0 devices is secured, the manual mechanism SHOULD BE used (configuring at least one T0 device manually).

Given the correct configuration of two T0 devices, maximally spaced on the fabric, the remaining intermediate systems calculate their tier number as follows:

- The local IS calculates an SPT (using SPF) setting the cost of every link to 1; this effectively calculates a topology only view of the network, without considering any configured link costs

- Ensure that at least two T0 are in the calculated SPT; otherwise abort

- Find the furthest T0; call this node A and set LD to the cost; the "farthest T0" is the T0 with the largest metric, or the farthest distance from the local calculating node

- Calculate an SPT (using SPF) from the perspective of A (above) setting the cost of every link to 1

- Find the furthest IS in A’s SPT; call this node B and set RD to the cost from A to B

- Calculate the tier number of the local IS by subtracting LD from RD

In the example network, assume 5A and 1C are manually configured as a T0, and are advertising their tier numbers. From here:

- From 1A the path to 5A is 4 hops; this is LD

- Run SPF from the perspective of 5A with all link metrics set to 1

- From 5A the path length to 1C is 4; this is RD

- RD - LD is 0 at 1A, so 1A is T0, or a ToR

This process will work for any spine and leaf fabric without "cross links."

5. Flooding Optimization

Flooding is perhaps the most challenging scaling issue for a link state protocol running on a dense, large scale fabric. To reduce the flooding of link state information in the form of Link State Protocol Data Units (LSPs), Openfabric takes advantage of information already available in the link state protocol, the list of the local intermediate system’s neighbor’s neighbors, and the fabric locality computed above. The following tables are required to compute a set of reflooders:

- Neighbor List (NL) list: The set of neighbors

- Neighbor’s Neighbors (NN) list: The set of neighbor’s neighbors; this can be calculated by running SPF truncated to two hops

- Do Not Reflood (DNR) list: The set of neighbors who should have LSPs (or fragments) who should not reflood LSPs

- Reflood (RF) list: The set of neighbors who should flood LSPs (or fragments) to their adjacent neighbors to ensure synchronization

NL is set to contain all neighbors, and sorted deterministically (for instance, from the highest IS identifier to the lowest). All intermediate systems within a single fabric SHOULD use the same mechanism for sorting the NL list. NN is set to contain all neighbor’s neighbors, or all intermediate systems that are two hops away, as determined by performing a truncated SPF. The DNR and RF tables are initially empty. To begin, the following steps are taken to reduce the size of NN and NL:

- Move any IS in NL with its tier (or fabric location) set to T0 to DNR

- Remove all intermediate systems from NL and NN that in the shortest path to the IS that originated the LSP

Then, for every IS in NL:

- If the current entry in NL is connected to any entries in NN:

  - Move the IS to RF

  - Remove the intermediate systems connected to the IS from NN

- Else move the IS to DNR

When flooding, LSPs transmitted to adjacent neighbors on the RF list will be transmitted normally. Adjacent intermediate systems on this list will reflood received LSPs into the next stage of the topology, ensuring database synchronization. LSPs transmitted to adjacent neighbors on the DNR list, however, MUST be transmitted using a circuit scope PDU as described in [RFC7356].

5.1. Flooding Failures

It is possible in some failure modes for flooding to be incomplete because of the flooding optimizations outlined. Specifically, if a reflooder fails, or is somehow disconnected from all the links across which it should be reflooding, it is possible an LSP is only partially flooded through the fabric. To prevent such situations, any IS receiving an LSP transmitted using DNR SHOULD:

- Set a short timer; the default should be less than one second
- When the timer expires, send a Complete Sequence Number Packet (CSNP) to all neighbors
- Process any Partial Sequence Number Packets (PSNPs) as required to resynchronize
- If a resynchronization is required, notify the network operator through a network management system

6. Other Optimizations

6.1. Transit Link Reachability

In order to reduce the amount of control plane state carried on large scale spine and leaf fabrics, openfabric implementations SHOULD NOT advertise reachability for transit links. These links MAY remain unnumbered, as IS-IS does not require layer 3 IP addresses to operate. Each IS SHOULD be configured with a single loopback address, which is assigned an IPv6 address, to provide reachability to intermediate systems which make up the fabric.

[RFC3277] SHOULD be supported on devices supporting openfabric with unnumbered interface in order to support traceability and network management.

6.2. Transiting T0 Intermediate Systems

In data center fabrics, ToR intermediate systems SHOULD NOT be used to transit between two T1 (or above) spine intermediate systems. The simplest way to prevent this is to set the overload bit [RFC3277] for all the LSPs originated from T0 intermediate systems. However, this solution would have the unfortunate side effect of causing all reachability beyond any T0 IS to have the same metric, and many implementations treat a set overload bit as a metric of 0xFFFF in calculating the Shortest Path Tree (SPT). This document proposes an alternate solution which preserves the leaf node metric, while still avoiding transiting T0 intermediate systems.

Specifically, all T0 intermediate systems SHOULD advertise their metric to reach any T1 adjacent neighbor with a cost of 0XFFE. T1 intermediate systems, on the other hand, will advertise T0 intermediate systems with the actual interface cost used to reach the T0 IS. Hence, links connecting T0 and T1 intermediate systems will be advertised with an asymmetric cost that discourages transiting T0 intermediate systems, while leaving reachability to the destinations attached to T0 devices the same.

7. Openfabric and Route Aggregation

While schemes may be designed so reachability information can be aggregated in Openfabric deployments, this is not a recommended configuraiton.

8. Security Considerations

This document outlines modifications to the IS-IS protocol for operation on large scale data center fabrics. While it does add new TLVs, and some local processing changes, it does not add any new security vulnerabilities to the operation of IS-IS. However, openfabric implementations SHOULD implement IS-IS cryptographic authentication, as described in [RFC5304], and should enable other security measures in accordance with best common practices for the IS-IS protocol.

If T0 intermediate systems are auto-detected using information outside Openfabric, it is possible to attack the calucations used for flooding reduction and auto-configuration of intermediate systems. For instance, if a request for an address pool is used as an indicator of an attached host, and hence receiving such a request causes an intermediate system to advertise itself as T0, it is possible for an attacker (or a simple mistake) to cause auto-configuration to fail. Any such auto-detection mechanims SHOULD BE secured using appropriate techniques, as described by any protocols or mechanisms used.

9. References

9.1. Normative References

[I-D.shen-isis-spine-leaf-ext]
Shen, N., Ginsberg, L., and S. Thyamagundalu, "IS-IS Routing for Spine-Leaf Topology", draft-shen-isis-spine-leaf-ext-05 (work in progress), January 2018.

[ISO10589]
International Organization for Standardization, "Intermediate system to Intermediate system intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode Network Service (ISO 8473)", ISO/IEC 10589:2002, Second Edition, Nov 2002.

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

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <https://www.rfc-editor.org/info/rfc2629>.

[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>.

[RFC5301] McPherson, D. and N. Shen, "Dynamic Hostname Exchange Mechanism for IS-IS", RFC 5301, DOI 10.17487/RFC5301, October 2008, <https://www.rfc-editor.org/info/rfc5301>.

[RFC5303] Katz, D., Saluja, R., and D. Eastlake 3rd, "Three-Way Handshake for IS-IS Point-to-Point Adjacencies", RFC 5303, DOI 10.17487/RFC5303, October 2008, <https://www.rfc-editor.org/info/rfc5303>.

[RFC5305] Li, T. and H. Smit, "IS-IS Extensions for Traffic Engineering", RFC 5305, DOI 10.17487/RFC5305, October 2008, <https://www.rfc-editor.org/info/rfc5305>.

[RFC5308] Hopps, C., "Routing IPv6 with IS-IS", RFC 5308, DOI 10.17487/RFC5308, October 2008, <https://www.rfc-editor.org/info/rfc5308>.

[RFC5309] Shen, N., Ed. and A. Zinin, Ed., "Point-to-Point Operation over LAN in Link State Routing Protocols", RFC 5309, DOI 10.17487/RFC5309, October 2008, <https://www.rfc-editor.org/info/rfc5309>.

[RFC5311] McPherson, D., Ed., Ginsberg, L., Previdi, S., and M. Shand, "Simplified Extension of Link State PDU (LSP) Space for IS-IS", RFC 5311, DOI 10.17487/RFC5311, February 2009, <https://www.rfc-editor.org/info/rfc5311>.

[RFC5316] Chen, M., Zhang, R., and X. Duan, "ISIS Extensions in Support of Inter-Autonomous System (AS) MPLS and GMPLS Traffic Engineering", RFC 5316, DOI 10.17487/RFC5316, December 2008, <https://www.rfc-editor.org/info/rfc5316>.

[RFC7356] Ginsberg, L., Previdi, S., and Y. Yang, "IS-IS Flooding Scope Link State PDUs (LSPs)", RFC 7356, DOI 10.17487/RFC7356, September 2014, <https://www.rfc-editor.org/info/rfc7356>.

[RFC7981] Ginsberg, L., Previdi, S., and M. Chen, "IS-IS Extensions for Advertising Router Information", RFC 7981, DOI 10.17487/RFC7981, October 2016, <https://www.rfc-editor.org/info/rfc7981>.

9.2. Informative References

[I-D.ietf-isis-segment-routing-extensions] Previdi, S., Ginsberg, L., Filsfils, C., Bashandy, A., Gredler, H., Litkowski, S., Decraene, B., and J. Tantsura, "IS-IS Extensions for Segment Routing", draft-ietf-isis-segment-routing-extensions-16 (work in progress), April 2018.

[I-D.ietf-spring-segment-routing] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-15 (work in progress), January 2018.

[RFC3277] McPherson, D., "Intermediate System to Intermediate System (IS-IS) Transient Blackhole Avoidance", RFC 3277, DOI 10.17487/RFC3277, April 2002, <https://www.rfc-editor.org/info/rfc3277>.

[RFC3719] Parker, J., Ed., "Recommendations for Interoperable Networks using Intermediate System to Intermediate System (IS-IS)", RFC 3719, DOI 10.17487/RFC3719, February 2004, <https://www.rfc-editor.org/info/rfc3719>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC5304] Li, T. and R. Atkinson, "IS-IS Cryptographic Authentication", RFC 5304, DOI 10.17487/RFC5304, October 2008, <https://www.rfc-editor.org/info/rfc5304>.

[RFC5440] Vasseur, JP., Ed. and JL. Le Roux, Ed., "Path Computation Element (PCE) Communication Protocol (PCEP)", RFC 5440, DOI 10.17487/RFC5440, March 2009, <https://www.rfc-editor.org/info/rfc5440>.

[RFC5449] Baccelli, E., Jacquet, P., Nguyen, D., and T. Clausen, "OSPF Multipoint Relay (MPR) Extension for Ad Hoc Networks", RFC 5449, DOI 10.17487/RFC5449, February 2009, <https://www.rfc-editor.org/info/rfc5449>.

[RFC5614] Ogier, R. and P. Spagnolo, "Mobile Ad Hoc Network (MANET) Extension of OSPF Using Connected Dominating Set (CDS) Flooding", RFC 5614, DOI 10.17487/RFC5614, August 2009, <https://www.rfc-editor.org/info/rfc5614>.

[RFC5837] Atlas, A., Ed., Bonica, R., Ed., Pignataro, C., Ed., Shen, N., and JR. Rivers, "Extending ICMP for Interface and Next-Hop Identification", RFC 5837, DOI 10.17487/RFC5837, April 2010, <https://www.rfc-editor.org/info/rfc5837>.

[RFC6232] Wei, F., Qin, Y., Li, Z., Li, T., and J. Dong, "Purge Originator Identification TLV for IS-IS", RFC 6232, DOI 10.17487/RFC6232, May 2011, <https://www.rfc-editor.org/info/rfc6232>.

[RFC7182] Herberg, U., Clausen, T., and C. Dearlove, "Integrity Check Value and Timestamp TLV Definitions for Mobile Ad Hoc Networks (MANETs)", RFC 7182, DOI 10.17487/RFC7182, April 2014, <https://www.rfc-editor.org/info/rfc7182>.

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

Authors’ Addresses

Russ White (editor)
LinkedIn

Email: russ@riw.us

Shawn Zandi (editor)
LinkedIn

Email: szandi@linkedin.com

i2rs Y. Zhuang, Ed.
Internet-Draft D. Shi
Intended status: Informational Huawei
Expires: March 4, 2018 R. Gu
China Mobile
August 31, 2017

YANG Data Model for Fabric Service delivery in Data Center Network
draft-zhuang-i2rs-dc-fabric-service-model-05

Abstract

This document defines a YANG data model that can be used to deliver fabric service for users within a data center network. This model is intended to be instantiated by management system. It provides an abstraction of services for a fabric network to be used by users. However it is not a configuration model used directly onto network infrastructures. It should be used combining with such as fabric topology data model defined in [I-D.zhuang-i2rs-yang-dc-fabric-network-topology] with specific fabric topology information to generate required configuration onto the related network elements to deliver the service.

Status of This Memo

This Internet-Draft will expire on March 4, 2018.

Copyright Notice

Table of Contents

1. Introduction

Network service provisioning is currently coupled with specific network topology and technology applied, which is technology and device oriented.

In the area of data center, this approach makes the management complex due to massive network devices involved and various applications deployed by multiple users (also known as tenants).

In the traditional way, the administrator has to be aware of the entire data center network before delivering services for users. When service request comes up, administrator has to divide the request into appropriate configurations and operations for all involved devices manually. Finally, these configurations are deployed onto network infrastructure, which requires personnel skills.

Actually different users share the same network infrastructure. A more dynamical way to deploy and manage the network is eager to be found out. Here we decompose the network management system into several layers in order to have network service provision more flexible and automatic. Each network layer is dedicated to be managed. What is more, all the layers can be combined to fulfill the delivery of the user’s service.

We can use three layers in data center network. The bottom one is physical infrastructure with massive devices. The middle one is fabric topology defined in [I-D.zhuang-i2rs-yang-dc-fabric-network-topology]. Unlike the physical layer, the fabric layer is used to display a fabric-based network view. In the fabric layer, a set of fabrics can exist with each managed independently. Furthermore, a bottom-up abstraction of fabric service is proposed to provide application centric interfaces facing to users which define network services regardless of beneath fabric topology and physical connections in the up layer.

This document defines a YANG [RFC6020] [RFC7950] data model focusing on the fabric service interfaces to define user fabric network services regardless of specific beneath network topology and devices. This model defines the generic configuration for fabric services within DC networks.

For example, this model can be used by the network orchestrator in which the fabric service interfaces are exposed. When a service from user application is requested, orchestrator adopts this model including service information and processes it into the topology layer through a DC controller. Thus a service is automatically and dynamically provided.

The service data model includes two main modules:

(a)Module "ietf-fabric-service" defines a module for user network service over fabric networks from the application centric view. To do so, it augments general network topology model defined in [I-D.ietf-i2rs-yang-network-topo] with logical components such as logical switches, logical routers as well as logical ports to carry network services requested by user applications.

(b)Module "ietf-fabric-endpoint" defines a module for hosts that run applications and generate traffics. The major usage of this module is to indicate the attachment points of a host in a user service network as well as in a physical network when it is initialed, so as to build bindings between physical layer and topology layer dynamically.

Besides, the model "ietf-fabric-topology" defined in [I-D.zhuang-i2rs-yang-dc-fabric-network-topology] with topology and resource as well as technology information is used to work together to implement configurations and operations of the fabric service onto the specific fabric infrastructure.

2. Concept and Terminology

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

2.1. Terminology

Fabric topology: a data center network can be decomposed to a set of fabric networks, while each of these fabrics composes a set of physical nodes/links of the physical infrastructure to form a fabric network. The fabric topology includes attributes of fabrics, such as gateway mode, involved nodes, roles of involved nodes etc al.

Fabric Service: it is used as a service interface of fabric networks to users, which uses logical elements to represent network connections between hosts for applications, regardless of a specific fabric topology deployment. Each service instance is based on a fabric topology, while a fabric can provide multiple service instances for different users, each of which is isolated to others.

Endpoint: an endpoint represents a host, which can be a virtual machine on a server or a bare-metal server.

Fabric capable device: a physical device (e.g. a switch) that supports fabric service and fabric topology models.

3. Fabric service framework overview

This draft provides a network service interface on top of fabrics network layer. Users can use these network service interfaces to deploy their applications over a data center network automatically and dynamically.

From the application centric point of view, user hosts can be considered to connect with other hosts through a switch if they are L2 reachable, alternatively, connect through a router if they are L3 reachable simply. So a user network can be abstracted into a logical network where L2 reachable represents logical switches connecting hosts and L3 reachable represents logical routers connecting switches.

With this concept, a user can use appropriate logical elements to define their networks and configure attributes of these elements such as vlan id, gateway etc al. All of these form a network service. For example, a fabric service diagram for a user is shown as below.

```
                        |C3 - L3 Interconnect
                        |
                        |logical port- external gateway
            +-----------|------------+
            |                        |
            |     Logical Router     |
            |                        |
            +---|--------------|-----+
                |              |logical port-gateway port
                |              |C2
                |              |
                |              |
    +-----------|----+       +-|--------------+
    |                |       |                | Logical port-Access port
    | Logical Switch |       | Logical Switch ------
    |                |       |                | C4 - L2 Interconnect
    +-|---------|----+       +-|---------|----+
      |         |              |C1       |
      |         |              |         |
+-----|---+  +--|------+  +----|----+  +-|-------+
| Endpoint|  | Endpoint|  | Endpoint|  | Endpoint|
+---------+  +---------+  +---------+  +---------+
```

Figure 1: Diagram of a fabric service

In the diagram, abstraction of network connections is focused as a very initial effort to abstract services for fabric-based DC networks. Based on the connection, we can add other network appliance for which the fabric service should be extended.

3.1. Service element

There are four major components regarding as service elements within a fabric service as depicted in Figure 1.

Logical Switch:

Works as a switch within a logical fabric network to provide L2 connections between hosts or to a logical router or to external networks. It can be bounded to one or several physical switches.

Logical Router:

Works as a router to provide L3 connections between logical switches or to external networks.

Logical Port:

Provides port function on logical switches and logical routers which claims their connections to others.

Endpoint:

Represents user hosts which can be a VM or a bare-metal server.

3.2. Functionality of connections

There are 4 connections between elements within the fabric service framework listed as follows:

C1: Endpoint attachment. It is used by an endpoint to connect to a logical switch.

C2: L2 to L3 attachment. Interface between a logical switch and a logical router within the same fabric.

C3: L3 interconnection which connects to a logical router.

C4: L2 interconnection which connects to a logical switch in another fabric.

Thinking of the functionality of different connections, a logical port can act as an access port (which provides C1/C4/C3 connection to a network element), or a service port (which provide C2 gateway connection) as shown in Figure 2.

```
                 +---------------+
                 |  Logical Port |
                 +--+----------+-+
                    |          |
                    |          |
                    |          |
                    |          |
     +--------------+-+      ''''''''''''''''''
     |  Access Port   |      '  Service Port  '
     +----------------+      '''|'''''''''''|''
                                |           |
                                |           |
                                |           |
                                |           |
                   +------------+--+     +--+------------+
                   | Gateway Port  |     |  External GW  |
                   +---------------+     +---------------+
```

Figure 2: Types of Logical port

When a logical port is noticed as an access port, there will be a corresponding physical port. In this situation, the required access configuration can be deployed on this physical port directly. However, there will be a gateway service if a logical port is noticed as a service port. In this situation, the management system should combine the gateway function and fabric territory at fabric topology layer together with the gateway configuration on the service port. By the combination, it is easy to figure out the appropriate devices in the physical infrastructure and their configurations for these devices respectively.

4. Fabric service model usage

4.1. Usage architecture

In section 3, a fabric service interface is provided for users to define their networks in a more concentrated and intuitive way. To be detailed, when a fabric service comes, the topology manager will parse services into configuration/operations onto specific devices automatically. In this process, service interface information and fabric topology information defined in [I-D.zhuang-i2rs-yang-dc-fabric-network-topology] is needed.

The whole process is shown in Fig.3. Fabric service module is used define network services for applications maybe by an orchestration for example, according to the topology architecture stated in [I-D.draft-ietf-i2rs-usecase-reqs-summary]. The topology information maintenance should be done by a topology manager. By combining information from different layers, a topology manager automatically generates configurations and operations of related devices and deploys them respectively over the physical fabric infrastructures.

```
                      +-------------------+
                      |                   |
                      |   Orchestrator    |
                      |                   |
                      +---------|---------+
                                |
                                | Fabric service
                                |
                                |
                      +---------V---------+
                      |                   |
                      | Topology Manager  |   Network Provider
                      |                   |
                      +--------^----------+
                               |
                               |
                               |Fabric Topology information
                               |
    ...........................|.............................
                               |
                         +----------------+
                        +----------------+|
                       +----------------+||
                       |                |||        Network
                       |    Device      ||+
                       |                |+
                       +----------------+
```

Figure 3: Fabric service Usage architecture

4.2. Multi-Layer interconnection

There are three layers in this usage.

At the service layer, a fabric service model is abstracted from fabric network used as an application-centric interface to define user networks. It focuses on the connection services from users' perspective. Using the fabric service interface, an administrator can define a logical network for each user over a single fabric network while each logical networks can be managed separately.

For the fabric topology layer, it collects and maintains the fabric topology information (including territory of the physical fabric, connections, gateway functions, roles of devices within the fabric and specific technologies for each fabric) upon the physical network layer.

With information provided by both fabric service as well as fabric topology, a fabric topology manager will calculates and generates configuration and operation for involved network devices in the physical layer so as to distribute and deploy them onto network infrastructure. The implementation of device configuration can be done in several ways, such as using defined data models for specific attributes, command lines. We will not limite any implemenation here.

The diagram of the management architecture and its relationship is depicted as below.

```
              +----------+
              |          |
              |   LR-1   |................
              |          |
              +--/---\---+
     gateway:   /     \   gateway:
    10.0.35.1  /       \ 10.0.36.1‘‘‘‘‘    Fabric Service Layer
              /         \             ‘
       +---------+   +---------+      ‘
       |         |   |         |      ‘
   +---| LSW-11  |   | LSW-12  |      ‘
   |   +-----|---+   +---------+      ‘
   |         |                |       ‘
   |         |                |       ‘              +--------+
   |         |                |       ‘              |        |
   |         |                |       ‘              |Fabric-3|
   |         |                |       ‘              |        |
   |         |                |       ‘              +-/ ---\-+
   |         |                |       ‘               /      \
   |         |                |       ‘              /        \
   |         |                |       ‘     +------ /+        +\ ------+
   |         |                |       ‘     |        |        |        |
   |         |                |       ‘     |Fabric-1|--------|Fabric-2|
   |         |                |       ‘     |        |        |        |
   |         |                |       ‘     +--------+        +--------+
   |         |                |       ‘‘‘‘‘gateway/roles
   |         +-----------------------+       of nodes
   |                          |      |           ‘ Fabric Topology Layer
   |              Fabric-1    |      |           ‘
   |              +-------+   |      |           ‘
   |            +--------+|‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘‘
```

```
   |           +--------+||   |      |
   |           |        ||+   |      |
   |           | SPINE  |+    |      |
   |           +--------+     |      |
   |              ||||        |      |
   |              |||-------+ |      |
   |             +|||------+| |      |    Physcial Layer
   |            +-||------+|| |      |
   |           +--|------+||+ |      |
   |          +---------+||+  |      |
   |          |  LEAF   ||+\  |      |
   |          | device  |+  \ |      |
   |          +---------+    \|      |
   |            /     \       \      |
   |           /       \      |\     |
   |          /         \     | \    |
   |        +/-+       +-\+   |  \ +-|+
   +--------|  |       |  |---+   \|  |
            +--+       +--+        +--+
     H1:10.0.35.2  H2:10.0.36.2  H3:10.0.35.3
```

Figure 4: Multi-layer interconnection

The mapping of nodes with access logical ports is realized by endpoints e.g. H1,H2 and H3 in Fig.4. An endpoint is instantiated by the orchestrator to indicate the locations of a host both in the logical layer as well as in the physical layer, so as to deliver services requested from the logical port onto the physical port in a dynamic manner. For H1 and H3, they are considered to connect to the same switch for user in the logical layer, even they attach to the different devices.

Besides, gateway configuration is defined at service layer while the gateway mode and gateway devices (for distributed gateway, the gateway should be deployed on LEAF devices, while for centralized gateway, the configuration should be on SPINE) are defined in fabric topology layer. By combing the gateway information from both layers, the system can automatically figure out the involved devices and generate appropriate configurations onto them.

5. Design of the data model

5.1. Fabric service module

As explained previously, network service for user network can be abstracted to sets of logical switches, logical routers and logical ports. Upon these logical elements, acl policies and gateway functions can be attached.

The fabric service module is defined by YANG module "ietf-fabric-service". The module is depicted in the following diagram.

```
 module: ietf-fabric-service
  augment /nw:networks/nw:network/nw:node:
    +--rw lsw-attribute
       +--rw lsw-uuid?     yang:uuid
       +--rw name?         string
       +--rw segment-id?   uint32
       +--rw network?      inet:ip-prefix
       +--rw external?     boolean
       +--rw fabric-acl* [fabric-acl-name]
          +--rw fabric-acl-name    string
  augment /nw:networks/nw:network/nw:node:
    +--rw lr-attribute
       +--rw lr-uuid?      yang:uuid
       +--rw name?         string
       +--rw vrf-ctx?      uint32
       +--rw fabric-acl* [fabric-acl-name]
       |  +--rw fabric-acl-name    string
       +--rw routes
          +--rw route* [destination-prefix]
             +--rw description?          string
             +--rw destination-prefix    inet:ipv4-prefix
             +--rw (next-hop-options)?
                +--:(simple-next-hop)
                   +--rw next-hop?             inet:ipv4-address
                   +--rw outgoing-interface?   nt:tp-id
  augment /nw:networks/nw:network/nw:node/nt:termination-point:
    +--rw lport-attribute
       +--rw lport-uuid?         yang:uuid
       +--rw name?               string
       +--rw port-layer
       |  +--rw layer-1-info
       |  |  +--rw location?   nt:tp-id
       |  +--rw layer-2-info
       |  |  +--rw access-type?      access-type
       |  |  +--rw access-segment?   uint32
       |  +--rw layer-3-info
       |     +--rw ip?               inet:ip-address
       |     +--rw network?          inet:ip-prefix
       |     +--rw mac?              yang:mac-address
       |     +--rw forward-enable?   boolean
       |     +--rw logical-switch?   nw:node-id
       +--rw fabric-acl* [fabric-acl-name]
```

```
          |  +--rw fabric-acl-name    string
          +--rw port-function
          |  +--rw (function-type)?
          |     +--:(ip-mapping)
          |        +--rw ip-mapping-entry* [external-ip]
          |           +--rw external-ip    inet:ipv4-address
          |           +--rw internal-ip?   inet:ipv4-address
          +--rw underlayer-ports* [port-ref]
             +--rw port-ref    instance-identifier
```

Figure 5: Fabric Service Module

To provide a logical network topology for DC fabric network, the module augments the original ietf-network and ietf-network-topology modules:

- New nodes for logical switch and logical router with additional data objects are introduced by augmenting the "node" list of the network module.
- Termination points for logical ports are augmented with logical port information and its reference to termination ports in the underlay topologies. As stated in section 3, the logical port may act as an access port which will be bounded to some physical port, or else it may be as a service point which connects to internal gateway or external gateway. Besides, it can also be attached with ACL rules.

5.2. Endpoint module

To represent user attachments points and map logical fabric configurations and operations of applications onto the physical fabric infrastructure, an endpoint is instantiated to represent a host of a user that runs applications.

The fabric endpoint module is defined by YANG module "ietf-fabric-endpoint". The module is depicted as follows:

```
module: ietf-fabric-endpoint
   +--rw endpoints
      +--rw endpoint* [endpoint-uuid]
         +--rw endpoint-uuid       yang:uuid
         +--rw own-fabric?         fabric:fabric-id
         +--rw mac-address?        yang:mac-address
         +--rw ip-address?         inet:ip-address
         +--rw gateway?            inet:ip-address
         +--rw public-ip?          inet:ip-address
         +--rw location
         |  +--rw node-ref?         fabrictype:node-ref
         |  +--rw tp-ref?           fabrictype:tp-ref
         |  +--rw access-type?      fabrictype:access-type
         |  +--rw access-segment?   uint32
         +--rw logical-location
            +--rw node-id?   nw:node-id
            +--rw tp-id?     nt:tp-id
```

Figure 6: Fabric endpoint module

By indicating locations of an endpoint in "location" container, the logical network elements such as logical nodes and logical termination points are bounded to the network elements in a specific fabric. Then the network configurations and operations from the logical network together with its belonged fabric topology information will further be distributed onto the bounding/related physical elements by the network topology manager.

Besides, the module defines three rpc commands to register, unregister and locate the endpoint onto both logical network and physical network shown as follows.

```
    rpcs:
       +---x register-endpoint
       |  +---w input
       |  |  +---w fabric-id?          fabric:fabric-id
       |  |  +---w endpoint-uuid?      yang:uuid
       |  |  +---w own-fabric?         fabric:fabric-id
       |  |  +---w mac-address?        yang:mac-address
       |  |  +---w ip-address?         inet:ip-address
       |  |  +---w gateway?            inet:ip-address
       |  |  +---w public-ip?          inet:ip-address
       |  |  +---w location
       |  |  |  +---w node-ref?         fabrictype:node-ref
       |  |  |  +---w tp-ref?           fabrictype:tp-ref
       |  |  |  +---w access-type?      fabrictype:access-type
       |  |  |  +---w access-segment?   uint32
       |  |  +---w logical-location
       |  |     +---w node-id?   nw:node-id
       |  |     +---w tp-id?     nt:tp-id
       |  +--ro output
       |     +--ro endpoint-id?   yang:uuid
       +---x unregister-endpoint
       |  +---w input
       |     +---w fabric-id?   fabric:fabric-id
       |     +---w ids*         yang:uuid
       +---x locate-endpoint
          +---w input
             +---w fabric-id?     fabric:fabric-id
             +---w endpoint-id?   yang:uuid
             +---w location
                +---w node-ref?         fabrictype:node-ref
                +---w tp-ref?           fabrictype:tp-ref
                +---w access-type?      fabrictype:access-type
                +---w access-segment?   uint32
```

Figure 7: Fabric endpoint module RPC

6. Fabric Service YANG Modules

```
<CODE BEGINS> file "ietf-fabric-service-types@2017-08-30.yang"
module ietf-fabric-service-types {

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

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

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

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

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

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

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

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

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

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

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


        revision "2017-08-30" {
                description
            "Initial revision of service types for fabric.";
        reference
                        "draft-zhuang-i2rs-dc-fabric-service-model-04";
    }


        ///groupings for logical element
        grouping logical-switch {
                description "grouping attributes for a logical switch.";

        leaf lsw-uuid {
            type yang:uuid;
                        description "logical switch id";
        }
        leaf name {
            type string;
                        description "logical switch name";
```

```
        }
        leaf segment-id {
            type uint32;
                        description "segement id";
        }
        leaf network {
            type inet:ip-prefix;
                        description "subnet";
        }
        leaf external {
            type boolean;
                        description "whether its a lsw to external network";
        }
        uses ft:acl-list;
    }

    grouping logical-router {
                description "grouping atttributes for a logical router";
        leaf lr-uuid {
            type yang:uuid;
                        description "logical router id";
        }
        leaf name {
            type string;
                        description "logical router name";
        }
        leaf vrf-ctx {
            type uint32;
                        description "logical router vrf id";
        }

        uses ft:acl-list;

                container routes {
                        description "routes";
                uses ft:route-group;
        }
    }

    grouping logical-port {
                description "grouping attributes for logical ports";
        leaf lport-uuid {
            type yang:uuid;
                        description "logical port id";
        }
        leaf name {
            type string;
                        description "logical port name";
```

```
        }
        container port-layer {
                        description "layer information of the lport";

            container layer-1-info {
                                description "layer 1 information of the lport";
                leaf location {
                    type nt:tp-id;
                                        description "L1 tp id";
                }
            }
            container layer-2-info {
                                description "layer 2 information of the lport";
                leaf access-type {
                    type ft:access-type;
                                        description "l2 access type";
                }
                leaf access-segment {
                    type uint32;
                                        description "access segement";
                }
            }
            container layer-3-info {
                                description "layer 3 information of the lport";
                leaf ip {
                    type inet:ip-address;
                                        description "ip address";
                }
                leaf network {
                    type inet:ip-prefix;
                                        description "ip prefix";
                }
                leaf mac {
                    type yang:mac-address;
                                        description "mac address";
                }
                leaf forward-enable {
                    type boolean;
                                        description "whether enable forward";
                }
                leaf logical-switch {
                    type nw:node-id;
                                        description "lsw id";
                }
            }
        }

        uses ft:acl-list;
```

```
        uses ft:port-functions;

        list underlayer-ports {
            key port-ref;
                        description "list of the corresponding underlay ports";
            leaf port-ref {
                type instance-identifier;
                                description "port reference";
            }
        }
    }
}
<CODE ENDS>

<CODE BEGINS> file "ietf-fabric-service@2017-08-30.yang"
module ietf-fabric-service {

    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-fabric-service";
    prefix fabric-services;

    import ietf-network { prefix nw;  }
    import ietf-network-topology { prefix nt;  }
        import ietf-fabric-service-types {prefix fst;}

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

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

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

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

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

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

    description
        "This module contains a collection of YANG definitions for Fabric servic
es.
                Copyright (c) 2016 IETF Trust and the persons identified as
                authors of the code.  All rights reserved.
```

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

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

        revision "2017-08-30" {
            description
                        "refer to fabric-service-type module instead of fabric-t
ype.";
                reference
                        "draft-zhuang-i2rs-yang-fabric-service-04";
        }

        revision "2017-03-03" {
            description
                        "remove rpc commands";
                reference
                        "draft-zhuang-i2rs-yang-fabric-service-01";
        }
    revision "2016-10-12" {
        description
            "Initial revision of fabric service.";
                reference
                        "draft-zhuang-i2rs-yang-fabric-service-00";
    }

   augment "/nw:networks/nw:network/nw:node" {
        description "Augmentation for logic switch nodes provided by fabrices.";

        container lsw-attribute {

                        description
                                "attributes for logical switches";
            uses fst:logical-switch;
        }
    }

    augment "/nw:networks/nw:network/nw:node" {
        description "Augmentation for logical router nodes provided by fabric se
rvices.";

        container lr-attribute {

                        description "attributes for logical routers";
```

```
            uses fst:logical-router;
        }
    }

    augment "/nw:networks/nw:network/nw:node/nt:termination-point" {
        description "Augmentation for logical port provided by fabric services."
;

        container lport-attribute {

                        description "attributes for logical ports";
            uses fst:logical-port;
        }
    }
}

<CODE ENDS>

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

    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-fabric-endpoint";
    prefix fabric-endpoints;

    import ietf-inet-types { prefix "inet"; revision-date "2013-07-15"; }
    import ietf-yang-types { prefix "yang"; revision-date "2013-07-15"; }
    import ietf-network { prefix nw;  }
    import ietf-network-topology { prefix nt;  }
    import ietf-fabric-types { prefix fabrictype;}
    import ietf-fabric-topology { prefix fabric; }

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

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

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

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

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

     Editor:    Danian Shi
```

```
                <mailto:shidanian@huawei.com>";


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

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

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

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

        revision "2017-06-29" {
                description
                        "compliant with NMDA";
                reference
                        "draft-zhuang-i2rs-yang-fabric-service-03";
        }

    revision "2016-10-12" {
        description
            "Initial revision of faas.";
                reference
                        "draft-zhuang-i2rs-yang-fabric-service-00";
    }

    grouping device-location {
        description "the location for this endponits in the physical network.";

        leaf node-ref {
            type fabrictype:node-ref;
                        description "node reference";
        }

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

        leaf access-type {
```

```
            type fabrictype:access-type;
            default "exclusive";
                        description "access type";
        }

        leaf access-segment {
            type uint32;
            default 0;
                        description "access segement";
        }
    }

    grouping endpoint-attributes {
                description "endpoint attributes";

        leaf endpoint-uuid {
            type yang:uuid;
                        description "endpoint id";
        }

        leaf own-fabric {
            type fabric:fabric-id;
                        description "fabric id";
        }

        leaf mac-address {
            type yang:mac-address;
                        description "mac addr";
        }

        leaf ip-address {
            type inet:ip-address;
                        description "ip addr";
        }

        leaf gateway {
            type inet:ip-address;
                        description "gateway ip";
        }

        leaf public-ip {
            type inet:ip-address;
                        description "public ip addr";
        }

        container location {
                        description "physical location of the endpoint";
            uses device-location;
```

```
        }

        container logical-location {
            description "The location for this endpoint in the logical network."
;

            leaf node-id {
                type nw:node-id;
                                description "node id";
            }

            leaf tp-id {
                type nt:tp-id;
                                description "port id";
            }
        }
    }

    container endpoints {
                description "endpoints registry for faas.";

        list endpoint {
            key "endpoint-uuid";
                        description "endpoint list";

                        uses endpoint-attributes;
        }
    }

    /*******************RPC***************************************/
    rpc register-endpoint {
        description
            "Register a new endpoing into the registry.";

        input {
           leaf fabric-id {
                type fabric:fabric-id;
                                description "fabric id";
            }

            uses endpoint-attributes;
        }
        output {
            leaf endpoint-id {
                type yang:uuid;
                                description "endpoint id";
            }
        }
    }
```

```
    rpc unregister-endpoint {
        description "Unregister an endpoint or endpoints from the registry.";
        input {
           leaf fabric-id {
                type fabric:fabric-id;
                                description "fabric id";
            }

            leaf-list ids {
                type yang:uuid;
                                description "a list of ids";
            }
        }
    }

    rpc locate-endpoint {
        description "Set the physical location of the endpoing.";
        input {
           leaf fabric-id {
                type fabric:fabric-id;
                                description "fabric id";
            }

            leaf endpoint-id {
                type yang:uuid;
                                description "endpoint id";
            }
            container location {
                        description "locations";
                uses device-location;
            }
        }
    }
}
<CODE ENDS>
```

7. Security Considerations

None.

8. IANA Considerations

None.

9. References

9.1. Normative References

[I-D.ietf-i2rs-yang-network-topo]
Clemm, A., Medved, J., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A Data Model for Network Topologies", draft-ietf-i2rs-yang-network-topo-14 (work in progress), June 2017.

[I-D.zhuang-i2rs-yang-dc-fabric-network-topology]
Zhuangyan, Z., Shi, D., Gu, R., and H. Ananthakrishnan, "A YANG Data Model for Fabric Topology in Data Center Network", draft-zhuang-i2rs-yang-dc-fabric-network-topology-04 (work in progress), July 2017.

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

[RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, DOI 10.17487/RFC2234, November 1997, <https://www.rfc-editor.org/info/rfc2234>.

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

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

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

9.2. 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.

Authors' Addresses

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

Email: zhuangyan.zhuang@huawei.com

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

Email: shidanian@huawei.com

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

Email: gurong_cmcc@outlook.com

I2RS Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

Y. Zhuang
D. Shi
Huawei
R. Gu
China Mobile
H. Ananthakrishnan
Packet Design
July 3, 2017

A YANG Data Model for Fabric Topology in Data Center Network
draft-zhuang-i2rs-yang-dc-fabric-network-topology-04

Abstract

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

Status of This Memo

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Table of Contents

1. Introduction

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

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

With this purpose, this document defines a YANG data model for the Fabric-based Data center topology by using YANG [6020][7950]. To do so, it augments the generic network and network topology data models defined in [I-D.ietf-i2rs-yang-network-topo] with information specific to Data Center fabric network.

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

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

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

2. Definitions an Acronyms

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

2.1. Terminology

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

2.2. Tree diagram

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

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

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

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

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

3. Model Overview

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

3.1. Topology Model structure

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

```
             +------------------------+
             |      network model     |
             +------------------------+
                          |
                          |
             +------------V-----------+
             | network topology model |
             +------------------------+
                          |
        +-----------+-----+------+-------------+
        |           |            |             |
    +---V----+  +---V----+   +---V----+   +----V---+
    |   L1   |  |   L2   |   |   L3   |   | Fabric |
    |topology|  |topology|   |topology|   |topology|
    |  model |  |  model |   |  model |   |  model |
    +--------+  +--------+   +--------+   +--------+
```

From the perspective of resource management and service provisioning for a Data Center network, the fabric topology model augments the basic network topology model with definitions and features specific to a DC fabric, to provide common configuration and operations for heterogeneous fabrics.

3.2. Fabric Topology Model

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

3.2.1. Fabric Topology

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

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

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

- o A new topology type "ietf-fabric-topology" is introduced and added under the "network-types" container of the ietf-network module.
- o Fabric is defined as a node under the network/node container. A new container of "fabric-attribute" is defined to carry attributes for a fabric network such as gateway mode, fabric types, involved device nodes and links etc al.
- o Termination points (in network topology module) are augmented with fabric port attributes defined in a container. The "termination-point" here can represent the "port" of a fabric that provides connections to other nodes, such as device internally, another fabric externally and also end hosts.

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

3.2.2. Fabric node extension

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

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

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

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

3.2.3. Fabric termination-point extension

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

As such, the "termination-point" in the fabric topology has three roles, including internal TP that connects to devices within a fabric, external TP that connects to outside network, as well as access TP to end hosts.

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

The structure of fabric ports is as follows:

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

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

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

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

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

4. Fabric YANG Module

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
    }

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

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

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

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

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

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

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

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

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

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

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

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

    grouping port-functions {
                description
                        "port functions";

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

                leaf fabric-id {
```

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

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

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

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

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

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

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

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

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

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

    }

        // augment statements

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

                uses fabric-network-type;
        }

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

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

            uses fabric-attributes;
        }
    }

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

        container fport-attribute {
```

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

5. Security Consideration

TBD

6. Acknowledgements

7. References

7.1. Normative References

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

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

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

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

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

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

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

7.2. Informative References

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

Appendix A. Non NMDA -state modules

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
        }

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

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

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

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

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

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

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

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

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

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

    }

        // augment statements

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

                uses fabric-network-type;
        }

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

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

            uses fabric-attributes;
        }
    }
}

                 <CODE ENDS>
```

Authors' Addresses

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

Email: zhuangyan.zhuang@huawei.com

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

Email: shidanian@huawei.com

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

Email: gurong_cmcc@outlook.com

Hariharan Ananthakrishnan
Packet Design

Email: hari@packetdesign.com