Network Working Group A. Bierman
Internet-Draft YumaWorks
Intended status: Standards Track M. Bjorklund
Expires: May 3, 2018 Tail-f Systems
K. Watsen
Juniper Networks
October 30, 2017

YANG Data Extensions
draft-bierman-netmod-yang-data-ext-01

Abstract

This document describes YANG mechanisms for defining abstract data structures with YANG. It is intended to replace and extend the "yang-data" extension statement defined in RFC 8040.

Table of Contents

1. Introduction

There is a need for standard mechanisms to allow the definition of abstract data that is not intended to be implemented as configuration or operational state. The "yang-data" extension statement from RFC 8040 [RFC8040] is defined for this purpose, however it is limited in its functionality.

The intended use of the "yang-data" extension is to model all or part of a protocol message, such as the "errors" definition in ietf-restconf.yang [RFC8040], or the contents of a file. However, protocols are often layered such that the header or payload portions of the message can be extended by external documents. The YANG statements that model a protocol need to support this extensibility that is already found in that protocol.

This document defines a new YANG extension statement called "augment-yang-data", which allows abstract data structures to be augmented from external modules, similar to the existing YANG "augment" statement. Note that "augment" cannot be used to augment a yang data structure since a YANG compiler or other tool is not required to understand the "yang-data" extension.

The "yang-data" extension from [RFC8040] has been copied here and updated to be more flexible. There is no longer a requirement for the "yang-data" statement to result in exactly one container object. There is no longer an assumption that a yang data structure can only be used as a top-level abstraction, instead of nested within some other data structure.

1.1. 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].

The following terms are used within this document:

- yang data structure: A data structure defined with the "yang-data" statement.

1.1.1. NMDA

The following terms are defined in the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores]. and are not redefined here:

- configuration

- operational state

1.1.2. YANG

The following terms are defined in [RFC7950]:

- absolute-schema-nodeid

- container

- data definition statement

- data node

- leaf

- leaf-list

- list

2. Definitions

2.1. Restrictions on Conceptual YANG Data

This document places restrictions on the "yang-data" external statements that can be used with the "yang-data" and "augment-yang-data" extensions. The conceptual data definitions are considered to be in the same identifier namespace as defined in section 6.2.1 of [RFC7950]. In particular, bullet 7:

> All leafs, leaf-lists, lists, containers, choices, rpcs, actions, notifications, anydatas, and anyxmls defined (directly or through a "uses" statement) within a parent node or at the top level of the module or its submodules share the same identifier namespace.

This means that conceptual data defined with the "yang-data" or "augment-yang-data" statements cannot have the same local-name as sibling nodes from regular YANG data definition statements or other "yang-data" or "augment-yang-data" statements.

This does not mean a yang data structure has to be used as a top-level protocol message or other top-level data structure. A yang data structure does not have to result in a single container.

2.2. YANG Data Extensions Module

The "yang-data-ext" module defines the "augment-yang-data" extension to augment conceptual data already defined with the "yang-data" extension. The RESTCONF "yang-data" extension has been moved to this document and updated.

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "yang-data-ext@2017-10-30.yang"

```
module yang-data-ext {
  // not assigning real module name and namespace unless
  // and until changed to a draft-ietf-netmod document
  namespace "urn:ietf:params:xml:ns:yang:yang-data-ext";
  prefix "yd";

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

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

```
        Author:   Andy Bierman
                  <mailto:andy@yumaworks.com>

        Author:   Martin Bjorklund
                  <mailto:mbj@tail-f.com>

        Author:   Kent Watsen
                  <mailto:kwatsen@juniper.net>";

     description
       "This module contains conceptual YANG specifications
        for defining abstract 'yang-data' data structures.

        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).";

     revision 2017-10-30 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: YANG Data Extensions.";
     }


     extension yang-data {
       argument name {
         yin-element true;
       }
       description
         "This extension is used to specify a YANG data template which
          represents conceptual data defined in YANG. It is
          intended to describe hierarchical data independent of
          protocol context or specific message encoding format.
          Data definition statements within a yang-data extension
          specify the generic syntax for the specific YANG data
          template, whose name is the argument of the yang-data
          extension statement.

          Note that this extension does not define a media-type.
          A specification using this extension MUST specify the
```

message encoding rules, including the content media type.

The mandatory 'name' parameter value identifies the YANG data template that is being defined. It contains the template name. This parameter is only used for readability purposes. There are no mechanisms to reuse yang-data by its template name value.

This extension is ignored unless it appears as a top-level statement. It MUST contain data definition statements that result in a set of data definition statements.

If the yang data template is intended to be used as a top-level structure, then the yang data template needs to result in a single container, so an instance of the YANG data template can thus be translated into an XML instance document, whose top-level element corresponds to the top-level container.

The module name and namespace value for the YANG module using the extension statement is assigned to each of the data definition statements resulting from the yang data template. The name of each data definition statement resulting from a yang data template is assigned to a top-level identifier name in the data node identifier namespace, according to RFC 7950, section 6.2.1.

The sub-statements of this extension MUST follow the 'data-def-stmt' rule in the YANG ABNF.

The XPath document root is the extension statement itself, such that the child nodes of the document root are represented by the data-def-stmt sub-statements within this extension. This conceptual document is the context for the following YANG statements:

- must-stmt
- when-stmt
- path-stmt
- min-elements-stmt
- max-elements-stmt
- mandatory-stmt
- unique-stmt
- ordered-by
- instance-identifier data type

The following data-def-stmt sub-statements are constrained when used within a yang-data-resource extension statement.

```
           - The list-stmt is not required to have a key-stmt defined.
           - The if-feature-stmt is ignored if present.
           - The config-stmt is ignored if present.
           - The available identity values for any 'identityref'
             leaf or leaf-list nodes is limited to the module
             containing this extension statement, and the modules
             imported into that module.
        ";
    }


    extension augment-yang-data {
      argument path {
        yin-element true;
      }
      description
        "This extension is used to specify an augmentation to
         conceptual data defined with the 'yang-data' statement.
         It is intended to describe hierarchical data independent
         of protocol context or specific message encoding format.

         This statement has almost the same structure as the
         'augment-stmt'. Data definition statements within this
         statement specify the semantics and generic syntax for the
         additional data to be added to the specific YANG data template,
         identified by the 'path' argument.

         The mandatory 'path' parameter value identifies the YANG
         conceptual data node that is being augmented, represented
         as an absolute-schema-nodeid string.

         This extension is ignored unless it appears as a top-level
         statement. The sub-statements of this extension MUST follow
         the 'data-def-stmt' rule in the YANG ABNF.

         The module name and namespace value for the YANG module using
         the extension statement is assigned to instance document data
         conforming to the data definition statements within
         this extension.

         The XPath document root is the augmented extension statement
         itself, such that the child nodes of the document root are
         represented by the data-def-stmt sub-statements within
         the augmented yang-data statement.

         The context node of the augment-yang-data statement is derived
         in the same way as the 'augment' statement, as defined in
         section 6.4.1 of [RFC7950]. This conceptual node is
```

```
         considered the context node for the following YANG statements:

           - must-stmt
           - when-stmt
           - path-stmt
           - min-elements-stmt
           - max-elements-stmt
           - mandatory-stmt
           - unique-stmt
           - ordered-by
           - instance-identifier data type

         The following data-def-stmt sub-statements are constrained
         when used within a augment-yang-data extension statement.

           - The list-stmt is not required to have a key-stmt defined.
           - The if-feature-stmt is ignored if present.
           - The config-stmt is ignored if present.
           - The available identity values for any 'identityref'
             leaf or leaf-list nodes is limited to the module
             containing this extension statement, and the modules
             imported into that module.

         Example:

            foo.yang {
               import ietf-restconf { prefix rc; }

               rc:yang-data foo-data {
                 container foo-con { }
               }
            }

            bar.yang {
               import yang-data-ext { prefix yd; }
               import foo { prefix foo; }

               yd:augment-yang-data /foo:foo-con {
                 leaf add-leaf1 { type int32; }
                 leaf add-leaf2 { type string; }
               }
            }
        ";
    }

  }

   <CODE ENDS>
```

3. IANA Considerations

3.1. YANG Module Registry

TBD

4. Security Considerations

This document defines YANG extensions that are used to define conceptual YANG data. It does not introduce any new vulnerabilities beyond those specified in YANG 1.1 [RFC7950].

5. Normative References

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-05 (work in progress), October 2017.

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

[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>.

[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>.

Appendix A. Change Log

A.1. v00 to v01

- o Added Martin and Kent as authors
- o Cloned and updated yang-data from RFC 8040
- o Added text to clarify that yang-data does not have to result in a single container

Appendix B. Open Issues

B.1. uses-yang-data

Is there a need for a separate grouping and uses mechanism for yang-data? Currently only real grouping-stmt and uses-stmt are used.

B.2. error-info

Is there a need for a special-purpose extension to define yang-data for the contents of the <error-info> node in NETCONF <rpc-error> and RESTCONF <errors> responses? This node is defined with anyxml so there is no way for a YANG tool to use real schema nodes, based on the RPC operation being requested or the error-app-tag that is being returned.

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 4, 2018

J. Clarke
B. Claise
Cisco Systems, Inc.

October 1, 2017

YANG module for yangcatalog.org
draft-clacla-netmod-model-catalog-02

Abstract

This document specifies a YANG module that contains metadata related to YANG modules and vendor implementations of those YANG modules.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC6020] [RFC7950] became the standard data modeling language of choice. Not only is it used by the IETF for specifying models, but also in many Standard Development Organizations (SDOs), consortia, and open-source projects: the IEEE, the Broadband Forum (BFF), DMTF, MEF, ITU, OpenDaylight, Open ROADM, Openconfig, sysrepo, and more.

With the rise of data model-driven management and the success of YANG as a key piece comes a challenge: the entire industry develops YANG models. In order for operators to automate coherent services, the industry must ensure the following:

1. Data models must work together

2. There exists a toolchain to help one search and understand models

3. Metadata is present to further describe model attributes

The site <https://www.yangcatalog.org> (and the YANG catalog that it provides) is an attempt to address these key tenants. From a high level point of view, the goal of this catalog is to become a reference for all YANG modules available in the industry, for both YANG developers (to search on what exists already) and for operators (to discover the more mature YANG models to automate services). This YANG catalog should not only contain pointers to the YANG modules themselves, but also contain metadata related to those YANG modules: What is the module type (service model or not?); what is the maturity level? (e.g., for the IETF: is this an RFC, a working group document or an individual draft?); is this module implemented?; who is the contact?; is there open-source code available? And we expect many more in the future. The industry has begun to understand that the metadata related to YANG models become equally important as the YANG models themselves.

This document defines a YANG [RFC6020] module called yang-catalog.yang that contains the metadata definitions that are complementary to the related YANG modules themselves. The design for this module is based on experience and real code. As such, it's expected that this YANG module will be a living document. Furthermore, new use cases, which require new metadata in this YANG module, are discovered on a regular basis.

The yangcatalog.org instantiation of the catalog provides a means for module authors and vendors implementing modules to upload their metadata, which is then searchable via an API, as well as using a variety of web-based tools. The instructions for contributing and searching for metadata can be found at <https://www.yangcatalog.org/contribe.php>.

1.1. Status of Work and Open Issues

The top open issues are:

1. Obtain feedback from vendors and SDOs

2. Socialize module at the IETF and incorporate feedback

3. Provide module bundle support

2. Learning from Experience

While implementing the catalog and tools at yangcatalog.org, we initially looked at the "Catalog and registry for YANG models" [I-D.openconfig-netmod-model-catalog] as a starting point but we quickly realized that the objectives are different. As a consequence, even if some of the information is similar, this YANG module started to diverge. Below are the justifications for the divergence, our observations, and our learning experience as we have been developing and getting feedback.

2.1. YANG Module Library

In order for the YANG catalog to become a complete inventory of which models are supported on the different platforms, content such as the support of the YANG module/deviation/feature/etc. should be easy to import and update. An easy way to populate this information is to have a similar structure as the YANG Module Library [RFC7895]. That way, querying the YANG Module Library from a platform provides, directly in the right format, the input for the YANG catalog inventory.

There are some similar entries between the YANG Module Library and the Openconfig catalog. For example, the Openconfig catalog model defines a "uri" leaf which is similar to "schema" from [RFC7895]). And this adds to the overall confusion.

2.2. YANG Catalog Data Model

The structure of the yang-catalog.yang module described in this document is found below:

```
module: yang-catalog
    +--rw catalog
       +--rw modules
       |  +--rw module* [name revision organization]
       |     +--rw name                        yang:yang-identifier
       |     +--rw revision                    union
       |     +--rw organization                string
       |     +--rw ietf
       |     |  +--rw ietf-wg?   string
       |     +--rw namespace                   inet:uri
       |     +--rw schema?                     inet:uri
       |     +--rw generated-from?             enumeration
       |     +--rw maturity-level?             enumeration
       |     +--rw document-name?              string
       |     +--rw author-email?               yc:email-address
       |     +--rw reference?                  inet:uri
       |     +--rw module-classification       enumeration
       |     +--rw compilation-status?         enumeration
       |     +--rw compilation-result?         inet:uri
       |     +--rw prefix?                     string
       |     +--rw yang-version?               enumeration
```

```
       |     +--rw description?               string
       |     +--rw contact?                   string
       |     +--rw module-type?               enumeration
       |     +--rw belongs-to?                yang:yang-identifier
       |     +--rw tree-type?                 enumeration
       |     +--rw submodule* [name revision]
       |     |  +--rw name        yang:yang-identifier
       |     |  +--rw revision    union
       |     |  +--rw schema?     inet:uri
       |     +--rw dependencies* [name]
       |     |  +--rw name        yang:yang-identifier
       |     |  +--rw revision?   union
       |     |  +--rw schema?     inet:uri
       |     +--rw dependents* [name]
       |     |  +--rw name        yang:yang-identifier
       |     |  +--rw revision?   union
       |     |  +--rw schema?     inet:uri
       |     +--rw semantic-version?          yc:semver
       |     +--rw derived-semantic-version?   yc:semver
       |     +--rw implementations
       |        +--rw implementation* [vendor platform software-version software
-flavor]
       |           +--rw vendor              string
       |           +--rw platform            string
       |           +--rw software-version    string
       |           +--rw software-flavor     string
       |           +--rw os-version?         string
       |           +--rw feature-set?        string
       |           +--rw os-type?            string
       |           +--rw feature*            yang:yang-identifier
       |           +--rw deviation* [name revision]
       |           |  +--rw name        yang:yang-identifier
       |           |  +--rw revision    union
       |           +--rw conformance-type?   enumeration
       +--rw vendors
          +--rw vendor* [name]
             +--rw name         string
             +--rw platforms
                +--rw platform* [name]
                   +--rw name                 string
                   +--rw software-versions
                      +--rw software-version* [name]
                         +--rw name                string
                         +--rw software-flavors
                            +--rw software-flavor* [name]
                               +--rw name         string
                               +--rw protocols
                               |  +--rw protocol* [name]
                               |     +--rw name                identityref
```

```
                              |     +--rw protocol-version*   string
                              |     +--rw capabilities*       string
                              +--rw modules
                                 +--rw module* [name revision organization]
                                    +--rw name                -> /catalog/modul
es/module/name
                                    +--rw revision            -> /catalog/modul
es/module/revision
                                    +--rw organization        -> /catalog/modul
es/module/organization
                                    +--rw os-version?         string
                                    +--rw feature-set?        string
                                    +--rw os-type?            string
                                    +--rw feature*            yang:yang-identif
ier
                                    +--rw deviation* [name revision]
                                    |  +--rw name        yang:yang-identifier
                                    |  +--rw revision    union
                                    +--rw conformance-type?   enumeration
```

Various elements of this module tree will be discussed in the subsequent sections.

2.3. Module Sub-Tree

Each module in the YANG Catalog is enumerated by its metadata and by various vendor implementations. While initially each module used the "module-list" grouping from the YANG Library [RFC7895], it was found that some of the nodes within that grouping such as "conformance-type", "feature", and "deviation" are only valid when a module is implemented by a server. As pure YANG data (which the Catalog is) it is not possible to provide meaningful values for those nodes. As such, common leafs were extracted from the YANG Library’s "module-list" for use in the module sub-tree of yang-catalog. Those server-specific nodes are moved under the implementation sub-tree. The yang-catalog module then augments these common nodes to add metadata elements that aid module developers and module consumers alike in understanding the relative maturity, compilation status, and the support contact(s) of each YANG module.

```
  +--rw modules
  |  +--rw module* [name revision organization]
  |     +--rw name                      yang:yang-identifier
  |     +--rw revision                  union
  |     +--rw organization              string
  |     +--rw ietf
  |     |  +--rw ietf-wg?   string
  |     +--rw namespace                 inet:uri
  |     +--rw schema?                   inet:uri
  |     +--rw generated-from?           enumeration
  |     +--rw maturity-level?           enumeration
  |     +--rw document-name?            string
```

```
     |      +--rw author-email?            yc:email-address
     |      +--rw reference?               inet:uri
     |      +--rw module-classification    enumeration
     |      +--rw compilation-status?      enumeration
     |      +--rw compilation-result?      inet:uri
     |      +--rw prefix?                  string
     |      +--rw yang-version?            enumeration
     |      +--rw description?             string
     |      +--rw contact?                 string
     |      +--rw module-type?             enumeration
     |      +--rw belongs-to?              yang:yang-identifier
     |      +--rw tree-type?               enumeration
     |      +--rw submodule* [name revision]
     |      |  +--rw name        yang:yang-identifier
     |      |  +--rw revision    union
     |      |  +--rw schema?     inet:uri
     |      +--rw dependencies* [name]
     |      |  +--rw name        yang:yang-identifier
     |      |  +--rw revision?   union
     |      |  +--rw schema?     inet:uri
     |      +--rw dependents* [name]
     |      |  +--rw name        yang:yang-identifier
     |      |  +--rw revision?   union
     |      |  +--rw schema?     inet:uri
     |      +--rw implementations
     |         +--rw implementation*
     |                 [vendor platform software-version software-flavor]
     |            +--rw vendor              string
     |            +--rw platform            string
     |            +--rw software-version    string
     |            +--rw software-flavor     string
     |            +--rw os-version?         string
     |            +--rw feature-set?        string
     |            +--rw os-type?            string
     |            +--rw feature*            yang:yang-identifier
     |            +--rw deviation* [name revision]
     |            |  +--rw name        yang:yang-identifier
     |            |  +--rw revision    union
     |            +--rw conformance-type?   enumeration
```

Many of these additional metadata fields are self-explanatory, especially given their descriptions in the module itself and the fact that many elements translate directly to YANG schema elements. However, those requiring additional explanation or context as to why they are needed are described in the subsequent sections.

2.4. Compilation Information

For the inventory to be complete, YANG modules at different stages of their lifecyle should be taken into account, including YANG modules that are clearly works-in-progress (i.e., that do not validate correctly either because of faulty YANG constructs, because of a faulty imported YANG module, or simply because of warnings). The results of compilation testing are denoted in the "compilation-status" leaf with links to the output of the tests stored in the "compilation-result" leaf. Note that some warnings seen in "compilation-result" are not always show-stoppers from a code generation point of view (see the Generated From section). Nonetheless, the compilation or validation status, along with the compilation output, provide a clear indication of a given YANG module’s development phase and stability. The current set of validator is pyang, confdc, yangdump-pro, and yanglint.

```
leaf compilation-status {
type enumeration {
enum passed {
  description
    "All compilers were able to compile this YANG module without
     any errors or warnings.";
}
enum passed-with-warnings {
  description
    "All compilers were able to compile this YANG module without
     any errors, but at least one of them caught a warning.";
}
enum failed {
  description
    "At least one of compilers found an error while
     compiling this YANG module.";
}
enum pending {
  description
    "The module was just added to the catalog and compilation testing is still
     in progress.";
}
enum unknown {
  description
    "There is not sufficient information about compilation status.  This Could
     mean compilation crashed causing it not to complete fully.";
}
}
description
"Status of the module, whether it was possible to compile this YANG module or
 there are still some errors/warnings.";
}
leaf compilation-result {
type string;
description
"Result of the compilation explaining specifically what error or warning occurre
d.
 This is not existing if compilation status is PASSED.";
}
```

The current instantiation of the YANG Catalog at <https://www.yangcatalog.org> uses a number of different YANG compilers for testing. The wrapper that handles validation attempts to use metadata from the catalog to determine which tests to perform on a given module. For example, if the module is authored by the IETF, IETF-specific tests will be conducted to provide the most accurate and complete set of tests possible.

2.5. Maturity Level

Models also have inherent maturity levels from their respective Standards Development Organizations (SDOs). These maturity levels help module consumers understand how complete, tested, etc. a module is.

```
leaf maturity-level {
type enumeration {
enum ratified {
  description
    "Maturity of a module that is fully approved (e.g., a standard).";
}
enum adopted {
  description
    "Maturity of a module that is actively being developed by a organization tow
ards ratification.";
}
enum initial {
  description
    "Maturity of a module that has been initially created, but has no official
     organization-level status.";
}
enum not-applicable {
  description
    "The maturity level is not used for vendor-supplied models, and thus all ven
dor
     modules will have a maturity of not-applicable";
}
}
description
"The current maturity of the module with respect to the body that created it.
 This allows one to understand where the module is in its overall life cycle.";
}
```

This enumeration mapping has been implemented for the YANG modules from IETF and BBF. The "maturity-level" MUST be "not-applicable" for all vendor-authored modules.

2.6. Generated From

While many models are written by hand (i.e., authored by humans) others are generated from things such as vendor code or CLI constructs or from SMI-based MIB modules. These "generated" modules do not necessarily require the same stringent validity checking that hand-written modules require. As such, these modules have a generated-from value that is designed to inform validators how much checking to do.

```
        leaf generated-from {
          type enumeration {
            enum "mib" {
              description
                "Module generated from Structure of Management Information (SMI)
                 MIB per RFC6643.";
            }
            enum "not-applicable" {
              description
                "Module was not generated but it was authored manually.";
            }
            enum "native" {
              description
                "Module generated from platform internal,
                 proprietary structure, or code.";
            }
          }
          default "not-applicable";
          description
            "This statement defines weather the module was generated or not.
             Default value is set to not-applicable, which means that module
             was created manualy and not generated.";
        }
```

2.7. Implementation

As of version 02 of openconfig-model-catalog.yang [I-D.openconfig-netmod-model-catalog] it is not possible to identify the implementations of one specific module. Instead modules are grouped into feature-bundle, and feature-bundles are implemented by devices. Because of this, we added our own implementation sub-tree under each module to yang-catalog.yang. Our implementation sub-tree is:

```
+--rw implementation* [vendor platform software-version software-flavor]
 +--rw vendor             string
 +--rw platform           string
 +--rw software-version   string
 +--rw software-flavor    string
 +--rw os-version?        string
 +--rw feature-set?       string
 +--rw os-type?           string
 +--rw feature*           yang:yang-identifier
 +--rw deviation* [name revision]
 |  +--rw name        yang:yang-identifier
 |  +--rw revision    union
 +--rw conformance-type?   enumeration
```

The keys in this sub-tree can be used in the "vendor" sub-tree defined below to walk through each vendor, platform, and software release to get a full list of supported YANG modules for that release.

The "software-flavor" key leaf identifies a variation of a specific version where YANG model support may be different. Depending on the vendor, this could be a license, additional software component, or a feature set.

The other non-key leaves in the implementation sub-tree represent optional elements of a software release that some vendors may choose to use for informational purposes. These leafs are duplicated under the vendor sub-tree.

2.8. Vendor Sub-Tree

The vendor sub-tree provides a way, especially for module consumers, to walk through a specific device and software release to find a list of modules supported therein. This sub-tree turns the "implementation" sub-tree on its head to provide an optimized index for one wanting to go from a platform to a full list of modules.

In addition to the module list, the vendor sub-tree lists the YANG-based protocols (e.g., NETCONF or RESTCONF) that the platforms support.

```
     +--rw vendors
        +--rw vendor* [name]
           +--rw name         string
           +--rw platforms
              +--rw platform* [name]
                 +--rw name                 string
                 +--rw software-versions
                    +--rw software-version* [name]
                       +--rw name                string
                       +--rw software-flavors
                          +--rw software-flavor* [name]
                             +--rw name         string
                             +--rw protocols
                             |  +--rw protocol* [name]
                             |     +--rw name                 identityref
                             |     +--rw protocol-version*   string
                             |     +--rw capabilities*       string
                             +--rw modules
                                +--rw module*
                                        [name revision organization]
                                   +--rw name                 leafref
                                   +--rw revision             leafref
                                   +--rw organization         leafref
                                   +--rw os-version?          string
                                   +--rw feature-set?         string
                                   +--rw os-type?             string
                                   +--rw feature*
                                   |       yang:yang-identifier
                                   +--rw deviation* [name revision]
                                   |  +--rw name
                                   |  |       yang:yang-identifier
                                   |  +--rw revision    union
                                   +--rw conformance-type?   enumeration
```

This sub-tree structure also enables one to look for YANG modules for a class of platforms (e.g., list of modules for Cisco, or list of modules for Cisco ASR9K routers) instead of only being able to look for YANG modules for a specific platform and software release.

2.9. Regex Expression Differences

Another challenge encountered when trying to using [I-D.openconfig-netmod-model-catalog] as the canonical catalog is the regular expression syntax it uses. The Openconfig module uses a POSIX-compliant regular expression syntax whereas YANG-based protocol implementations like ConfD [1] expect the IETF-chosen W3C syntax. In order to load the Openconfig catalog in such engines, changes to the regular expression syntax had to be done, and these one-off changes are not supportable.

3. YANG Catalog Use Cases

The YANG Catalog module is currently targeted to address the following use cases.

3.1. YANG Search Metadata

The yangcatalog.org toolchain provides a service for searching [2] for YANG modules based on keywords. The resulting search data currently stores the module and node metadata in a proprietary format along with the search index data. By populating the yang-catalog module, this search service can instead pull the metadata from the implementation of the module. Populating this instance of the yang-catalog module will be using an API that is still under development, but will ultimately allow SDOs and vendors to provide metadata and ensure the search service has the most up-to-date data for all available modules.

3.2. Identify YANG Module Support in Devices

By organizing the yang-catalog module so that one can either find all implementations for a given module, or find all modules supported by a vendor platform and software release, the catalog will provide a straight-forward way for one to understand the extent of YANG module support in participating vendors’ software releases. Eventually a web-based graphical interface will be connected to this on yangcatalog.org to make it easier for consumers to leverage the instance of the yang-catalog module for this use case.

3.3. Identify The Backward Compatibility between YANG Module Revisions

The YANG catalog contains not only the most up-to-date YANG module revision of a given module, but keeps all previous revisions as well. With APIs in mind, it’s important to understand whether different YANG module revisions are backward compatible (this is specifically imported for native YANG modules, i.e. the ones where generated-from = native). This document uses the following semver.org semantic [semver] to compare the YANG module backwards (in)compatibility:

MAJOR is incremented when the new version of the specification is incompatible with previous versions.

MINOR is incremented when new functionality is added in a manner that is backward-compatible with previous versions.

PATCH is incremented when bug fixes are made in a backward-compatible manner.

Two distinct leaves in the YANG module contains this semver semantic:

the semantic-version leaf contains the value reported as metadata by a specific YANG module.

the derived-semantic-version leaf is established by examining the the YANG module themselves. As such, only the YANG syntax, as opposed to the implementation changes that lead some some semantic changes.

Typically, an Openconfig YANG module would contain an extension, which is mapped to the semantic-version leaf.

```
     // extension statements
     extension openconfig-version {
       argument "semver" {
         yin-element false;
       }
       description
         "The OpenConfig version number for the module. This is
         expressed as a semantic version number of the form:
           x.y.z
          where:
           * x corresponds to the major version,
           * y corresponds to a minor version,
           * z corresponds to a patch version.
         This version corresponds to the model file within which it is
         defined, and does not cover the whole set of OpenConfig models.
         Where several modules are used to build up a single block of
         functionality, the same module version is specified across each
         file that makes up the module.

         A major version number of 0 indicates that this model is still
         in development (whether within OpenConfig or with industry
         partners), and is potentially subject to change.

         Following a release of major version 1, all modules will
         increment major revision number where backwards incompatible
         changes to the model are made.

         The minor version is changed when features are added to the
         model that do not impact current clients use of the model.

         The patch-level version is incremented when non-feature changes
         (such as bugfixes or clarifications to human-readable
         descriptions that do not impact model functionality) are made
         that maintain backwards compatibility.

         The version number is stored in the module meta-data.";
     }
```

Note that the absolute numbers in the semantic-version and derived-semantic-version are actually meaningless: the difference between two YANG module semver fields should be looked at.

4. YANG Catalog YANG module

The structure of the model defined in this document is described by the YANG module below.

<CODE BEGINS> file "yang-catalog@2017-09-26.yang"

```
  module yang-catalog {
    namespace "urn:ietf:params:xml:ns:yang:yang-catalog";
    prefix yc;

    import ietf-yang-types {
      prefix yang;
    }
    import ietf-yang-library {
      prefix yanglib;
    }
    import ietf-inet-types {
      prefix inet;
    }

    organization
      "yangcatalog.org";
    contact
      "Benoit Claise <bclaise@cisco.com>

       Joe Clarke <jclarke@cisco.com>";
    description
      "This module contains metadata pertinent to each YANG module, as
       well as a list of vendor implementations for each module.  The
       structure is laid out in such a way as to make it possible to
       locate metadata and vendor implementation on a per-module basis
       as well as obtain a list of available modules for a given
       vendor's platform and specific software release.";

    revision 2017-09-26 {
      description
        "* Add leafs for tracking dependencies and dependents
         * Simply the generated-from enumerated values
         * Refine the type for compilation-result to be an inet:uri
         * Add leafs for semantic versioning";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-08-18 {
      description
        "* Reorder organization to be with the other module keys
         * Add a belongs-to leaf to track a submodule's parent";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-28 {
      description
        "* Revert config false nodes as we need to be able to set these via <edi
t-config>

         * Make conformance-type optional as not all vendors implement yang-libr
ary
```

```
         * Re-add the path typedef";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-26 {
      description
        "A number of improvements based on YANG Doctor review:

         * Remove references to 'server' in leafs describing YANG data
         * Fold the augmentation module leafs directly under /catalog/modules/mo
dule
         * Use identities for protocols instead of an emumeration
         * Make some extractable fields 'config false'
         * Fix various types
         * Normalize enums to be lowercase
         * Add a leaf for module-classification
         * Change yang-version to be an enum
         * Add module conformance, deviation and feature leafs under the impleme
ntation branches";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-14 {
      description
        "Modularize some of the leafs and create typedefs so they
         can be shared between the API input modules.";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-03 {
      description
        "Initial revision.";
      reference
        "
         YANG Catalog <https://yangcatalog.org>";
    }

    /*
     * Identities
     */

    identity protocol {
      description
        "Abstract base identity for a YANG-based protocol.";
    }

    identity netconf {
      base protocol;
      description
        "Protocol identity for NETCONF as described in RFC 6241.";
    }

    identity restconf {
```

```
      base protocol;
      description
        "Protocol identity for RESTCONF as described in RFC 8040.";
    }

    /*
     * Typedefs
     */

    typedef email-address {
      type string {
        pattern "[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+([.][a-zA-Z0-9-]+
)*";
      }
      description
        "This type represents a string with an email address.";
    }

    typedef path {
      type string {
        pattern "([A-Za-z]:|[\\w-]+(\\.[\\w-]+)*)?(([/\\\\][\\w@.-]+)+)";
      }
      description
        "This type represents a string with path to the file.";
    }

    typedef semver {
      type string {
        pattern "[0-9]+\\.[0-9]+\\.[0-9]+";
      }
      description
        "A semantic version in the format of x.y.z, where:

         x = the major version number
         y = the minor version number
         z = the patch version number

         Changes to the major version number denote backwards-incompatible
         changes between two revisions of the same module.

         Changes to the minor version number indicate there have been new
         backwards-compatible features introduced in the later version of
         a module.

         Changes to the patch version indicate bug fixes between two
         versions of a module.";
      reference "Semantic Versioning 2.0.0 <http://semver.org/>";
    }
```

```
    container catalog {
      description
        "Root container of yang-catalog holding two main branches -
         modules and vendors. The modules sub-tree contains all the modules in
         the catalog and all of their metadata with their implementations.
         The vendor sub-tree holds modules for specific vendors, platforms,
         software-versions, and software-flavors. It contains reference to a
         name and revision of the module in order to reference the module’s full
         set of metadata.";
      container modules {
        description
          "Container holding the list of modules";
        list module {
          key "name revision organization";
          description
            "Each entry represents one revision of one module
             for one organization.";
          uses yang-lib-common-leafs;
          leaf organization {
            type string;
            description
              "This statement defines the party responsible for this
               module.  The argument is a string that is used to specify a textu
al
               description of the organization(s) under whose auspices this modu
le
               was developed.";
          }
          uses organization-specific-metadata;
          leaf namespace {
            type inet:uri;
            mandatory true;
            description
              "The XML namespace identifier for this module.";
          }
          uses yang-lib-schema-leaf;
          uses catalog-module-metadata;
          list submodule {
            key "name revision";
            description
              "Each entry represents one submodule within the
               parent module.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          list dependencies {
            key "name";
            description
              "Each entry represents one dependency.";
            uses yang-lib-common-leafs;
```

```
            uses yang-lib-schema-leaf;
          }
          list dependents {
            key "name";
            description
              "Each entry represents one dependent.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          leaf semantic-version {
            type yc:semver;
            description
              "The formal semantic version of a module as provided by the module
               itself.  If the module does not provide a semantic version, this
leaf
               will not be specified.";
          }
          leaf derived-semantic-version {
            type yc:semver;
            description
              "The semantic version of a module as compared to other revisions o
f
               the same module.  This value is computed algorithmically by order
ing
               all revisions of a given module and comparing them to look for ba
ckwards
               incompatible changes.";
          }
          container implementations {
            description
              "Container holding lists of per-module implementation details.";
            list implementation {
              key "vendor platform software-version software-flavor";
              description
                "List of module implementations.";
              leaf vendor {
                type string;
                description
                  "Organization that implements this module.";
              }
              leaf platform {
                type string;
                description
                  "Platform on which this module is implemented.";
              }
              leaf software-version {
                type string;
                description
                  "Name of the version of software.  With respect to most networ
k device appliances,
                   this will be the operating system version.  But for other YAN
G module
                   implementation, this would be a version of appliance software
.  Ultimately,
                   this should correspond to a version string that will be recog
nizable by
```

```
                   the consumers of the platform.";
              }
              leaf software-flavor {
                type string;
                description
                  "A variation of a specific version where
                   YANG model support may be different.  Depending on the vendor
, this could
                   be a license, additional software component, or a feature set
.";
              }
              uses shared-implementation-leafs;
              uses yang-lib-imlementation-leafs;
            }
          }
        }
      }
      container vendors {
        description
          "Container holding lists of organizations that publish YANG modules.";
        list vendor {
          key "name";
          description
            "List of organizations publishing YANG modules.";
          leaf name {
            type string;
            description
              "Name of the maintaining organization -- the name should be
               supplied in the official format used by the organization.
               Standards Body examples:
                 IETF, IEEE, MEF, ONF, etc.
               Commercial entity examples:
                 AT&T, Facebook, <Vendor>
               Name of industry forum examples:
               OpenConfig, OpenDaylight, ON.Lab";
          }
          container platforms {
            description
              "Container holding list of platforms.";
            list platform {
              key "name";
              description
                "List of platforms under specific vendor";
              leaf name {
                type string;
                description
                  "Name of the platform";
              }
              container software-versions {
                description
```

```
                  "Container holding list of versions of software versions.";
                list software-version {
                  key "name";
                  description
                    "List of version of software versions under specific vendor,
 platform.";
                  leaf name {
                    type string;
                    description
                      "Name of the version of software.  With respect to most ne
twork device appliances,
                       this will be the operating system version.  But for other
 YANG module
                       implementation, this would be a version of appliance soft
ware.  Ultimately,
                       this should correspond to a version string that will be r
ecognizable by
                       the consumers of the platform.";
                  }
                  container software-flavors {
                    description
                      "Container holding list of software flavors.";
                    list software-flavor {
                      key "name";
                      description
                        "List of software flavors under specific vendor, platfor
m, software-version.";
                      leaf name {
                        type string;
                        description
                          "A variation of a specific version where
                           YANG model support may be different.  Depending on th
e vendor, this could
                           be a license, additional software component, or a fea
ture set.";
                      }
                      container protocols {
                        description
                          "List of the protocols";
                        list protocol {
                          key "name";
                          description
                            "YANG-based protocol that is used on the device.  Ne
w identities
                             are expected to be added to address other YANG-base
d protocols.";
                          leaf name {
                            type identityref {
                              base yc:protocol;
                            }
                            description
                              "Identity of the YANG-based protocol that is suppo
rted.";
                          }
                          leaf-list protocol-version {
                            type string;
                            description
                              "Version of the specific protocol.";
                          }
```

```
                          leaf-list capabilities {
                            type string;
                            description
                              "Listed name of capabilities that are
                               supported by the specific device.";
                          }
                        }
                      }
                      container modules {
                        description
                          "Container holding list of modules.";
                        list module {
                          key "name revision organization";
                          description
                            "List of references to YANG modules under specific v
endor, platform, software-version,
                             software-flavor.  Using these references, the compl
ete set of metadata can be
                             retrieved for each module.";
                          leaf name {
                            type leafref {
                              path "/catalog/modules/module/name";
                            }
                            description
                              "Reference to a name of the module that is contain
ed in specific vendor, platform,
                               software-version, software-flavor.";
                          }
                          leaf revision {
                            type leafref {
                              path "/catalog/modules/module/revision";
                            }
                            description
                              "Reference to a revision of the module that is con
tained in specific vendor,
                               platform, software-version, software-flavor.";
                          }
                          leaf organization {
                            type leafref {
                              path "/catalog/modules/module/organization";
                            }
                            description
                              "Reference to the authoring organization of the mo
dule for the implemented
                                module.";
                          }
                          uses shared-implementation-leafs;
                          uses yang-lib-imlementation-leafs;
                        }
                      }
                    }
                  }
                }
```

```
              }
            }
          }
        }
      }
    }

    grouping catalog-module-metadata {
      uses shared-module-leafs;
      leaf compilation-status {
        type enumeration {
          enum "passed" {
            description
              "All compilers were able to compile this YANG module without
               any errors or warnings.";
          }
          enum "passed-with-warnings" {
            description
              "All compilers were able to compile this YANG module without
               any errors, but at least one of them caught a warning.";
          }
          enum "failed" {
            description
              "At least one of compilers found an error while
               compiling this YANG module.";
          }
          enum "pending" {
            description
              "The module was just added to the catalog and compilation testing
is still
               in progress.";
          }
          enum "unknown" {
            description
              "There is not sufficient information about compilation status.  Th
is Could
               mean compilation crashed causing it not to complete fully.";
          }
        }
        description
          "Status of the module, whether it was possible to compile this YANG mo
dule or
           there are still some errors/warnings.";
      }
      leaf compilation-result {
        type inet:uri;
        description
          "Link to the result of the compilation explaining specifically what er
ror or
           warning occurred.  This is not existing if compilation status is PASS
ED.";
      }
      leaf prefix {
```

```
        type string;
        description
          "Statement of yang that is used to define the prefix associated with
           the module and its namespace. The prefix statement’s argument is
           the prefix string that is used as a prefix to access a module. The
           prefix string MAY be used to refer to definitions contained in the
           module, e.g., if:ifName.";
      }
      leaf yang-version {
        type enumeration {
          enum "1.0" {
            description
              "YANG version 1.0 as defined in RFC 6020.";
          }
          enum "1.1" {
            description
              "YANG version 1.1 as defined in RFC 7950.";
          }
        }
        description
          "The optional yang-version statement specifies which version of the
           YANG language was used in developing the module.";
      }
      leaf description {
        type string;
        description
          "This statement takes as an argument a string that
           contains a human-readable textual description of this definition.
           The text is provided in a language (or languages) chosen by the
           module developer; for the sake of interoperability, it is RECOMMENDED
           to choose a language that is widely understood among the community of
           network administrators who will use the module.";
      }
      leaf contact {
        type string;
        description
          "This statement provides contact information for the module.
           The argument is a string that is used to specify contact information
           for the person or persons to whom technical queries concerning this
           module should be sent, such as their name, postal address, telephone
           number, and electronic mail address.";
      }
      leaf module-type {
        type enumeration {
          enum "module" {
            description
              "If YANG file contains module.";
          }
```

```
          enum "submodule" {
            description
              "If YANG file contains sub-module.";
          }
        }
        description
          "Whether a file contains a YANG module or sub-module.";
      }
      leaf belongs-to {
        when "../module-type = 'submodule'" {
          description
            "Include the module's parent when it is a submodule.";
        }
        type yang:yang-identifier;
        description
          "Name of the module that includes this submodule.";
      }
      leaf tree-type {
        type enumeration {
          enum "split" {
            description
              "This module uses a split config/operational state layout.";
          }
          enum "nmda-compatible" {
            description
              "This module is compatible with the Network Management Datastores
               Architecture (NMDA) and combines config and operational state nod
es.";
          }
          enum "transitional-extra" {
            description
              "This module is derived as a '-state' module to allow for transiti
oning
               to a full NMDA-compliant tree structure.";
          }
          enum "openconfig" {
            description
              "This module uses the Openconfig data element layout.";
          }
          enum "unclassified" {
            description
              "This module does not belong to any category or can't be determine
d.";
          }
          enum "not-applicable" {
            description
              "This module is not applicable. For example, because the YANG modu
le only contains typedefs, groupings, or is a submodule";
          }
        }
        description
          "The type of data element tree used by the module as it relates to the
```

```
           Network Management Datastores Architecture.";
        reference "draft-dsdt-nmda-guidelines Guidelines for YANG Module Authors
 (NMDA)";
      }
      description
        "Grouping of YANG module metadata that extends the common list defined i
n the YANG
         Module Library (RFC 7895).";
    }

    grouping organization-specific-metadata {
      container ietf {
        when "../organization = 'ietf'" {
          description
            "Include this container specific metadata of the IETF.";
        }
        leaf ietf-wg {
          type string;
          description
            "Working group that authored the document containing this module.";
        }
        description
          "Include this container for the IETF-specific organization metadata.";
      }
      description
        "Any organization that has some specific metadata of the yang module and
 want them add to the
         yang-catalog, should augment this grouping. This grouping is for any me
tadata that can't be used for
         every yang module.";
    }

    grouping yang-lib-common-leafs {
      leaf name {
        type yang:yang-identifier;
        description
          "The YANG module or submodule name.";
      }
      leaf revision {
        type union {
          type yanglib:revision-identifier;
          type string {
            length "0";
          }
        }
        description
          "The YANG module or submodule revision date.
           A zero-length string is used if no revision statement
           is present in the YANG module or submodule.";
      }
      description
        "The YANG module or submodule revision date.
```

```
          A zero-length string is used if no revision statement
          is present in the YANG module or submodule.";
      reference "RFC7895 YANG Module Library : common-leafs grouping";
    }

    grouping yang-lib-schema-leaf {
      leaf schema {
        type inet:uri;
        description
          "Contains a URL that represents the YANG schema
           resource for this module or submodule.
           This leaf will only be present if there is a URL
           available for retrieval of the schema for this entry.";
      }
      description
        "These are a subset of leafs from the yang-library (RFC 7895) that provi
de some
         extractable fields for catalog modules.  The module-list grouping canno
t be
         used from yang-library as modules themselves cannot have conformance wi
thout
         a server.";
      reference "RFC7895 YANG Module Library : schema-leaf grouping";
    }

    grouping yang-lib-imlementation-leafs {
      leaf-list feature {
        type yang:yang-identifier;
        description
          "List of YANG feature names from this module that are
           supported by the server, regardless of whether they are
           defined in the module or any included submodule.";
      }
      list deviation {
        key "name revision";
        description
          "List of YANG deviation module names and revisions
           used by this server to modify the conformance of
           the module associated with this entry.  Note that
           the same module can be used for deviations for
           multiple modules, so the same entry MAY appear
           within multiple 'module' entries.
           The deviation module MUST be present in the 'module'
           list, with the same name and revision values.
           The 'conformance-type' value will be 'implement' for
           the deviation module.";
        uses yang-lib-common-leafs;
      }
      leaf conformance-type {
        type enumeration {
          enum "implement" {
```

```
            description
              "Indicates that the server implements one or more
               protocol-accessible objects defined in the YANG module
               identified in this entry.  This includes deviation
               statements defined in the module.
               For YANG version 1.1 modules, there is at most one
               module entry with conformance type 'implement' for a
               particular module name, since YANG 1.1 requires that,
               at most, one revision of a module is implemented.
               For YANG version 1 modules, there SHOULD NOT be more
               than one module entry for a particular module name.";
          }
          enum "import" {
            description
              "Indicates that the server imports reusable definitions
               from the specified revision of the module but does
               not implement any protocol-accessible objects from
               this revision.
               Multiple module entries for the same module name MAY
               exist.  This can occur if multiple modules import the
               same module but specify different revision dates in
               the import statements.";
          }
        }
        // Removing the mandatory true for now as not all vendors may have
        // this information if they do not implement yang-library.
        //mandatory true;
        description
          "Indicates the type of conformance the server is claiming
           for the YANG module identified by this entry.";
      }
      description
        "This is a set of leafs extracted from the yang-library that are
         specific to server implementations.";
      reference "RFC7895 YANG Module Library : module-list grouping";
    }

    grouping shared-implementation-leafs {
      leaf os-version {
        type string;
        description
          "Version of the operating system using this module.  This is primarily
 useful if
           the software implementing the module is an application that requires
a specific
           operating system.";
      }
      leaf feature-set {
        type string;
        description
```

```
          "An optional feature of the software that is required in order to impl
ement this
           module.  Some form of this must be incorporated in software-version o
r
           software-flavor, but can be broken out here for additional clarity.";
      }
      leaf os-type {
        type string;
        description
          "Type of the operating system using this module.  This is primarily us
eful if
           the software implementing the module is an application that requires
a
           specific operating system.";
      }
      description
        "Grouping of non-key leafs to be used in the module and vendor sub-trees
.";
    }

    grouping shared-module-leafs {
      leaf generated-from {
        type enumeration {
          enum "mib" {
            description
              "Module generated from Structure of Management Information (SMI)
               MIB per RFC6643.";
          }
          enum "not-applicable" {
            description
              "Module was not generated but it was authored manually.";
          }
          enum "native" {
            description
              "Module generated from platform internal,
               proprietary structure, or code.";
          }
        }
        default "not-applicable";
        description
          "This statement defines weather the module was generated or not.
           Default value is set to not-applicable, which means that module
           was created manualy and not generated.";
      }
      leaf maturity-level {
        type enumeration {
          enum "ratified" {
            description
              "Maturity of a module that is fully approved (e.g., a standard).";
          }
          enum "adopted" {
            description
              "Maturity of a module that is actively being developed by a organi
zation towards ratification.";
```

```
          }
          enum "initial" {
            description
              "Maturity of a module that has been initially created, but has no
official
               organization-level status.";
          }
          enum "not-applicable" {
            description
              "The maturity level is not used for vendor-supplied models, and th
us all vendor
               modules will have a maturity of not-applicable";
          }
        }
        description
          "The current maturity of the module with respect to the body that crea
ted it.
           This allows one to understand where the module is in its overall life
 cycle.";
      }
      leaf document-name {
        type string;
        description
          "The name of the document from which the module was extracted or taken
;
           or that provides additional context about the module.";
      }
      leaf author-email {
        type yc:email-address;
        description
          "Contact email of the author who is responsible for this module.";
      }
      leaf reference {
        type inet:uri;
        description
          "A string that is used to specify a textual cross-reference to an exte
rnal document, either
           another module that defines related management information, or a docu
ment that provides
           additional information relevant to this definition.";
      }
      leaf module-classification {
        type enumeration {
          enum "network-service" {
            description
              "Network Service YANG Module that describes the configuration, sta
te
               data, operations, and notifications of abstract representations o
f
               services implemented on one or multiple network elements.";
          }
          enum "network-element" {
            description
              "Network Element YANG Module that describes the configuration, sta
te
               data, operations, and notifications of specific device-centric
               technologies or features.";
          }
```

```
          enum "unknown" {
            description
              "In case that there is not sufficient information about how to cla
ssify the module.";
          }
          enum "not-applicable" {
            description
              "The YANG module abstraction type is neither a Network Service YAN
G Module
               nor a Network Element YANG Module.";
          }
        }
        mandatory true;
        description
          "The high-level classification of the given YANG module.";
        reference "RFC8199 YANG Module Classification";
      }
      description
        "These leafs are shared among the yang-catalog and its API.";
    }

    grouping online-source-file {
      leaf owner {
        type string;
        mandatory true;
        description
          "Username or ID of the owner of the version control system repository.
";
      }
      leaf repository {
        type string;
        mandatory true;
        description
          "The name of the repository.";
      }
      leaf path {
        type yc:path;
        mandatory true;
        description
          "Location within the repository of the module file.";
      }
      leaf branch {
        type string;
        description
          "Revision control system branch or tag to use to find the module.  If
this is not
           specified, the head of the repository is used.";
      }
      description
        "Networked version control system location of the module file.";
    }
  }
```

<CODE ENDS>

5. Security Considerations

The goal of the YANG Catalog module and yangcatalog.org is to document a large library of YANG modules and their implementations. Already, we have seen some SDOs hestitant to provide modules that have not reached a "ratified" maturity level because of intellectual property leakage concerns or simply organization process that mandates only fully ratified modules can be published. Care must be paid that through private automated testing and validation of such modules that their metadata does not leak before the publishing organization approves the release of such data.

Similarly, from a vendor implementation standpoint, data that is exposed to the catalog before the vendor has fully vetted it could cause confusion amongst that vendor’s customers or reveal product releases to the market before they have been officially announced.

Ultimately, there is a balance to be struck with respect to providing a rich library of YANG module metadata, and doing so at the right time to avoid information leakage.

6. IANA Considerations

No IANA action is requested.

7. References

7.1. Normative References

[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>.

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

[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>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

[semver] "Semantic Versioning 2.0.0", <https://www.semver.org>.

7.2. Informative References

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

[I-D.openconfig-netmod-model-catalog]
Shaikh, A., Shakir, R., and K. D'Souza, "Catalog and registry for YANG models", draft-openconfig-netmod-model-catalog-02 (work in progress), March 2017.

7.3. URIs

[1] https://developer.cisco.com/site/confD/index.gsp

[2] https://www.yangcatalog.org/yang-search

Appendix A. Acknowledgments

The authors would like to thanks Miroslav Kovac for this help on this YANG module and the yangcatalog.org implementation. We would also like to thank Radek Krejci for his extensive review and suggestions for improvement.

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

Appendix B. Changes From Previous Revisions

RFC Editor to remove this section prior to publication.

Draft -00 to -01:

- o Redesign of module sub-tree based on review.
- o Modularize some leafs and create typedefs to share with API YANG modules.
- o Add module conformance-type, deviation and feature leafs under the implementation branch.
- o Change yang-version to be an enum.
- o Add a leaf for module-classification based on [RFC8199].
- o Normalize enums to be lowercase.
- o Use identities for protocols instead of an enumeration.
- o Make conformance-type optional as not all vendors implement [RFC7895].
- o Add a leaf for tree-type based on [I-D.ietf-netmod-revised-datastores].
- o Add a reference to contributing to the YANG Catalog at yangcatalog.org.
- o Various wording and style changes to the document text.

Draft -01 to -02:

- o Add a belongs-to leaf to track parent modules.
- o Add leafs to track dependents and dependencies for a given module.
- o Simplify the generated-from enumerated values.
- o Refine the type for compilation-result to be an inet:uri.
- o Add leafs for semantic versioning.
- o Reorder the organization leaf to be with other module keys.
- o Add text to describe generated-from and semantic versioning.

Authors' Addresses

Joe Clarke
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Network Working Group   J. Clarke
Internet-Draft   B. Claise
Intended status: Standards Track   Cisco Systems, Inc.
Expires: October 5, 2018   April 3, 2018

YANG module for yangcatalog.org
draft-clacla-netmod-model-catalog-03

Abstract

This document specifies a YANG module that contains metadata related to YANG modules and vendor implementations of those YANG modules.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC6020] [RFC7950] became the standard data modeling language of choice. Not only is it used by the IETF for specifying models, but also in many Standard Development Organizations (SDOs), consortia, and open-source projects: the IEEE, the Broadband Forum (BFF), DMTF, MEF, ITU, OpenDaylight, Open ROADM, Openconfig, sysrepo, and more.

With the rise of data model-driven management and the success of YANG as a key piece comes a challenge: the entire industry develops YANG models. In order for operators to automate coherent services, the industry must ensure the following:

1. Data models must work together

2. There exists a toolchain to help one search and understand models

3. Metadata is present to further describe model attributes

The site <https://www.yangcatalog.org> (and the YANG catalog that it provides) is an attempt to address these key tenants. From a high level point of view, the goal of this catalog is to become a reference for all YANG modules available in the industry, for both YANG developers (to search on what exists already) and for operators (to discover the more mature YANG models to automate services). This YANG catalog should not only contain pointers to the YANG modules themselves, but also contain metadata related to those YANG modules: What is the module type (service model or not?); what is the maturity level? (e.g., for the IETF: is this an RFC, a working group document or an individual draft?); is this module implemented?; who is the contact?; is there open-source code available? And we expect many more in the future. The industry has begun to understand that the metadata related to YANG models become equally important as the YANG models themselves.

This document defines a YANG [RFC7950] module called yang-catalog.yang that contains the metadata definitions that are complementary to the related YANG modules themselves. The design for this module is based on experience and real code. As such, it’s expected that this YANG module will be a living document. Furthermore, new use cases, which require new metadata in this YANG module, are discovered on a regular basis.

The yangcatalog.org instantiation of the catalog provides a means for module authors and vendors implementing modules to upload their metadata, which is then searchable via an API, as well as using a variety of web-based tools. The instructions for contributing and searching for metadata can be found at <https://www.yangcatalog.org/contribe.php>.

1.1. Status of Work and Open Issues

The top open issues are:

1. Obtain feedback from vendors and SDOs

2. Socialize module at the IETF and incorporate feedback

3. Provide module bundle support

2. Learning from Experience

While implementing the catalog and tools at yangcatalog.org, we initially looked at the "Catalog and registry for YANG models" [I-D.openconfig-netmod-model-catalog] as a starting point but we quickly realized that the objectives are different. As a consequence, even if some of the information is similar, this YANG module started to diverge. Below are the justifications for the divergence, our observations, and our learning experience as we have been developing and getting feedback.

2.1. YANG Module Library

In order for the YANG catalog to become a complete inventory of which models are supported on the different platforms, content such as the support of the YANG module/deviation/feature/etc. should be easy to import and update. An easy way to populate this information is to have a similar structure as the YANG Module Library [RFC7895]. That way, querying the YANG Module Library from a platform provides, directly in the right format, the input for the YANG catalog inventory.

There are some similar entries between the YANG Module Library and the Openconfig catalog. For example, the Openconfig catalog model defines a "uri" leaf which is similar to "schema" from [RFC7895]). And this adds to the overall confusion.

2.2. YANG Catalog Data Model

The structure of the yang-catalog.yang module described in this document is found below. The meaning of the symbols in this and seubsequent tree diagrams in this document is explained in [I-D.ietf-netmod-yang-tree-diagrams]:

```
module: yang-catalog
    +--rw catalog
       +--rw modules
       |  +--rw module* [name revision organization]
       |     +--rw name                         yang:yang-identifier
       |     +--rw revision                     union
       |     +--rw organization                 string
       |     +--rw ietf
       |     |  +--rw ietf-wg?   string
       |     +--rw namespace                    inet:uri
       |     +--rw schema?                      inet:uri
       |     +--rw generated-from?              enumeration
       |     +--rw maturity-level?              enumeration
       |     +--rw document-name?               string
       |     +--rw author-email?                yc:email-address
       |     +--rw reference?                   inet:uri
       |     +--rw module-classification        enumeration
       |     +--rw compilation-status?          enumeration
       |     +--rw compilation-result?          inet:uri
```

```
        |     +--rw prefix?                     string
        |     +--rw yang-version?               enumeration
        |     +--rw description?                string
        |     +--rw contact?                    string
        |     +--rw module-type?                enumeration
        |     +--rw belongs-to?                 yang:yang-identifier
        |     +--rw tree-type?                  enumeration
        |     +--rw yang-tree?                  inet:uri
        |     +--rw expires?                    yang:date-and-time
        |     +--rw expired?                    union
        |     +--rw submodule* [name revision]
        |     |  +--rw name        yang:yang-identifier
        |     |  +--rw revision    union
        |     |  +--rw schema?     inet:uri
        |     +--rw dependencies* [name]
        |     |  +--rw name        yang:yang-identifier
        |     |  +--rw revision?   union
        |     |  +--rw schema?     inet:uri
        |     +--rw dependents* [name]
        |     |  +--rw name        yang:yang-identifier
        |     |  +--rw revision?   union
        |     |  +--rw schema?     inet:uri
        |     +--rw semantic-version?           yc:semver
        |     +--rw derived-semantic-version?   yc:semver
        |     +--rw implementations
        |        +--rw implementation* [vendor platform software-version software
-flavor]
        |           +--rw vendor             string
        |           +--rw platform           string
        |           +--rw software-version   string
        |           +--rw software-flavor    string
        |           +--rw os-version?        string
        |           +--rw feature-set?       string
        |           +--rw os-type?           string
        |           +--rw feature*           yang:yang-identifier
        |           +--rw deviation* [name revision]
        |           |  +--rw name        yang:yang-identifier
        |           |  +--rw revision    union
        |           +--rw conformance-type?   enumeration
        +--rw vendors
           +--rw vendor* [name]
              +--rw name         string
              +--rw platforms
                 +--rw platform* [name]
                    +--rw name                string
                    +--rw software-versions
                       +--rw software-version* [name]
                          +--rw name                string
                          +--rw software-flavors
```

```
                            +--rw software-flavor* [name]
                               +--rw name         string
                               +--rw protocols
                               |  +--rw protocol* [name]
                               |     +--rw name                identityref
                               |     +--rw protocol-version*   string
                               |     +--rw capabilities*       string
                               +--rw modules
                                  +--rw module* [name revision organization]
                                     +--rw name                -> /catalog/modul
es/module/name

                                     +--rw revision            -> deref(../name)
/../revision
                                     +--rw organization        -> deref(../revis
ion)/../organization
                                     +--rw os-version?         string
                                     +--rw feature-set?        string
                                     +--rw os-type?            string
                                     +--rw feature*            yang:yang-identif
ier
                                     +--rw deviation* [name revision]
                                     |  +--rw name        yang:yang-identifier
                                     |  +--rw revision    union
                                     +--rw conformance-type?   enumeration
```

Various elements of this module tree will be discussed in the subsequent sections.

2.3. Module Sub-Tree

Each module in the YANG Catalog is enumerated by its metadata and by various vendor implementations. While initially each module used the "module-list" grouping from the YANG Library [RFC7895], it was found that some of the nodes within that grouping such as "conformance-type", "feature", and "deviation" are only valid when a module is implemented by a server. As pure YANG data (which the Catalog is) it is not possible to provide meaningful values for those nodes. As such, common leafs were extracted from the YANG Library’s "module-list" for use in the module sub-tree of yang-catalog. Those server-specific nodes are moved under the implementation sub-tree. The yang-catalog module then augments these common nodes to add metadata elements that aid module developers and module consumers alike in understanding the relative maturity, compilation status, and the support contact(s) of each YANG module.

```
     +--rw modules
     |  +--rw module* [name revision organization]
     |     +--rw name                     yang:yang-identifier
     |     +--rw revision                 union
     |     +--rw organization             string
     |     +--rw ietf
     |     |  +--rw ietf-wg?   string
```

```
   |     +--rw namespace                inet:uri
   |     +--rw schema?                  inet:uri
   |     +--rw generated-from?          enumeration
   |     +--rw maturity-level?          enumeration
   |     +--rw document-name?           string
   |     +--rw author-email?            yc:email-address
   |     +--rw reference?               inet:uri
   |     +--rw module-classification    enumeration
   |     +--rw compilation-status?      enumeration
   |     +--rw compilation-result?      inet:uri
   |     +--rw prefix?                  string
   |     +--rw yang-version?            enumeration
   |     +--rw description?             string
   |     +--rw contact?                 string
   |     +--rw module-type?             enumeration
   |     +--rw belongs-to?              yang:yang-identifier
   |     +--rw tree-type?               enumeration
   |     +--rw yang-tree?               inet:uri
   |     +--rw expires?                 yang:date-and-time
   |     +--rw expired?                 union
   |     +--rw submodule* [name revision]
   |     |  +--rw name        yang:yang-identifier
   |     |  +--rw revision    union
   |     |  +--rw schema?     inet:uri
   |     +--rw dependencies* [name]
   |     |  +--rw name        yang:yang-identifier
   |     |  +--rw revision?   union
   |     |  +--rw schema?     inet:uri
   |     +--rw dependents* [name]
   |     |  +--rw name        yang:yang-identifier
   |     |  +--rw revision?   union
   |     |  +--rw schema?     inet:uri
   |     +--rw implementations
   |        +--rw implementation*
   |                [vendor platform software-version software-flavor]
   |           +--rw vendor                 string
   |           +--rw platform               string
   |           +--rw software-version       string
   |           +--rw software-flavor        string
   |           +--rw os-version?            string
   |           +--rw feature-set?           string
   |           +--rw os-type?               string
   |           +--rw feature*               yang:yang-identifier
   |           +--rw deviation* [name revision]
   |           |  +--rw name        yang:yang-identifier
   |           |  +--rw revision    union
   |           +--rw conformance-type?   enumeration
```

Many of these additional metadata fields are self-explanatory, especially given their descriptions in the module itself and the fact that many elements translate directly to YANG schema elements. However, those requiring additional explanation or context as to why they are needed are described in the subsequent sections.

2.4. Compilation Information

For the inventory to be complete, YANG modules at different stages of their lifecyle should be taken into account, including YANG modules that are clearly works-in-progress (i.e., that do not validate correctly either because of faulty YANG constructs, because of a faulty imported YANG module, or simply because of warnings). The results of compilation testing are denoted in the "compilation-status" leaf with links to the output of the tests stored in the "compilation-result" leaf. Note that some warnings seen in "compilation-result" are not always show-stoppers from a code generation point of view (see the Generated From section). Nonetheless, the compilation or validation status, along with the compilation output, provide a clear indication of a given YANG module’s development phase and stability. The current set of validator is pyang, confdc, yangdump-pro, and yanglint.

```
leaf compilation-status {
type enumeration {
enum passed {
  description
    "All compilers were able to compile this YANG module without
     any errors or warnings.";
}
enum passed-with-warnings {
  description
    "All compilers were able to compile this YANG module without
     any errors, but at least one of them caught a warning.";
}
enum failed {
  description
    "At least one of compilers found an error while
     compiling this YANG module.";
}
enum pending {
  description
    "The module was just added to the catalog and compilation testing is still
     in progress.";
}
enum unknown {
  description
    "There is not sufficient information about compilation status.  This Could
     mean compilation crashed causing it not to complete fully.";
}
}
description
"Status of the module, whether it was possible to compile this YANG module or
 there are still some errors/warnings.";
}
leaf compilation-result {
type string;
description
"Result of the compilation explaining specifically what error or warning occurre
d.
 This is not existing if compilation status is PASSED.";
}
```

The current instantiation of the YANG Catalog at <https://www.yangcatalog.org> uses a number of different YANG compilers for testing. The wrapper that handles validation attempts to use metadata from the catalog to determine which tests to perform on a given module. For example, if the module is authored by the IETF, IETF-specific tests will be conducted to provide the most accurate and complete set of tests possible.

2.5. Maturity Level

Models also have inherent maturity levels from their respective Standards Development Organizations (SDOs). These maturity levels help module consumers understand how complete, tested, etc. a module is.

```
leaf maturity-level {
type enumeration {
enum ratified {
  description
    "Maturity of a module that is fully approved (e.g., a standard).";
}
enum adopted {
  description
    "Maturity of a module that is actively being developed by a organization tow
ards ratification.";
}
enum initial {
  description
    "Maturity of a module that has been initially created, but has no official
     organization-level status.";
}
enum not-applicable {
  description
    "The maturity level is not used for vendor-supplied models, and thus all ven
dor
     modules will have a maturity of not-applicable";
}
}
description
"The current maturity of the module with respect to the body that created it.
 This allows one to understand where the module is in its overall life cycle.";
}
```

This enumeration mapping has been implemented for the YANG modules from IETF and BBF. The "maturity-level" MUST be "not-applicable" for all vendor-authored modules.

In addition to a module’s maturity, modules that are part of works-in-progress (e.g., IETF internet drafts) may expire if work ceases on the related document. To track that, the catalog has two module leafs: "expires" and "expired". The "expires" leaf indicates a date and time when the module is expected to expire whereas the "expired" leaf indicates whether or not the module has already expired. For those modules that will never expire, the "expired" leaf MUST be set to "not-applicable".

2.6. Generated From

While many models are written by hand (i.e., authored by humans) others are generated from things such as vendor code or CLI constructs or from SMI-based MIB modules. These "generated" modules do not necessarily require the same stringent validity checking that hand-written modules require. As such, these modules have a generated-from value that is designed to inform validators how much checking to do.

```
leaf generated-from {
  type enumeration {
    enum "mib" {
      description
        "Module generated from Structure of Management Information (SMI)
         MIB per RFC6643.";
    }
    enum "not-applicable" {
      description
        "Module was not generated but it was authored manually.";
    }
    enum "native" {
      description
        "Module generated from platform internal,
         proprietary structure, or code.";
    }
  }
  default "not-applicable";
  description
    "This statement defines weather the module was generated or not.
     Default value is set to not-applicable, which means that module
     was created manualy and not generated.";
}
```

2.7. Implementation

As of version 02 of openconfig-model-catalog.yang [I-D.openconfig-netmod-model-catalog] it is not possible to identify the implementations of one specific module. Instead modules are grouped into feature-bundle, and feature-bundles are implemented by devices. Because of this, we added our own implementation sub-tree under each module to yang-catalog.yang. Our implementation sub-tree is:

```
+--rw implementation* [vendor platform software-version software-flavor]
 +--rw vendor              string
 +--rw platform            string
 +--rw software-version    string
 +--rw software-flavor     string
 +--rw os-version?         string
 +--rw feature-set?        string
 +--rw os-type?            string
 +--rw feature*            yang:yang-identifier
 +--rw deviation* [name revision]
 |  +--rw name        yang:yang-identifier
 |  +--rw revision    union
 +--rw conformance-type?   enumeration
```

The keys in this sub-tree can be used in the "vendor" sub-tree defined below to walk through each vendor, platform, and software release to get a full list of supported YANG modules for that release.

The "software-flavor" key leaf identifies a variation of a specific version where YANG model support may be different. Depending on the vendor, this could be a license, additional software component, or a feature set.

The other non-key leaves in the implementation sub-tree represent optional elements of a software release that some vendors may choose to use for informational purposes. These leafs are duplicated under the vendor sub-tree.

2.8. Vendor Sub-Tree

The vendor sub-tree provides a way, especially for module consumers, to walk through a specific device and software release to find a list of modules supported therein. This sub-tree turns the "implementation" sub-tree on its head to provide an optimized index for one wanting to go from a platform to a full list of modules.

In addition to the module list, the vendor sub-tree lists the YANG-based protocols (e.g., NETCONF or RESTCONF) that the platforms support.

```
  +--rw vendors
     +--rw vendor* [name]
        +--rw name         string
        +--rw platforms
           +--rw platform* [name]
              +--rw name                 string
              +--rw software-versions
                 +--rw software-version* [name]
                    +--rw name                string
                    +--rw software-flavors
                       +--rw software-flavor* [name]
                          +--rw name         string
                          +--rw protocols
                          |  +--rw protocol* [name]
                          |     +--rw name                identityref
                          |     +--rw protocol-version*   string
                          |     +--rw capabilities*       string
                          +--rw modules
                             +--rw module*
                                     [name revision organization]
                                +--rw name                leafref
                                +--rw revision            leafref
                                +--rw organization        leafref
                                +--rw os-version?         string
                                +--rw feature-set?        string
                                +--rw os-type?            string
                                +--rw feature*
                                |       yang:yang-identifier
                                +--rw deviation* [name revision]
                                |  +--rw name
                                |  |       yang:yang-identifier
                                |  +--rw revision    union
                                +--rw conformance-type?   enumeration
```

This sub-tree structure also enables one to look for YANG modules for a class of platforms (e.g., list of modules for Cisco, or list of modules for Cisco ASR9K routers) instead of only being able to look for YANG modules for a specific platform and software release.

2.9. Regex Expression Differences

Another challenge encountered when trying to using [I-D.openconfig-netmod-model-catalog] as the canonical catalog is the regular expression syntax it uses. The Openconfig module uses a POSIX-compliant regular expression syntax whereas YANG-based protocol implementations like ConfD [1] expect the IETF-chosen W3C syntax. In order to load the Openconfig catalog in such engines, changes to the regular expression syntax had to be done, and these one-off changes are not supportable.

3. YANG Catalog Use Cases

The YANG Catalog module is currently targeted to address the following use cases.

3.1. YANG Search Metadata

The yangcatalog.org toolchain provides a service for searching [2] for YANG modules based on keywords. The resulting search data currently stores the module and node metadata in a proprietary format along with the search index data. By populating the yang-catalog module, this search service can instead pull the metadata from the implementation of the module. Populating this instance of the yang-catalog module will be using an API that is still under development, but will ultimately allow SDOs and vendors to provide metadata and ensure the search service has the most up-to-date data for all available modules.

3.2. Identify YANG Module Support in Devices

By organizing the yang-catalog module so that one can either find all implementations for a given module, or find all modules supported by a vendor platform and software release, the catalog will provide a straight-forward way for one to understand the extent of YANG module support in participating vendors' software releases. Eventually a web-based graphical interface will be connected to this on yangcatalog.org to make it easier for consumers to leverage the instance of the yang-catalog module for this use case.

3.3. Identify The Backward Compatibility between YANG Module Revisions

The YANG catalog contains not only the most up-to-date YANG module revision of a given module, but keeps all previous revisions as well. With APIs in mind, it's important to understand whether different YANG module revisions are backward compatible (this is specifically imported for native YANG modules, i.e. the ones where generated-from = native). This document uses the following semver.org semantic [semver] to compare the YANG module backwards (in)compatibility:

MAJOR is incremented when the new version of the specification is incompatible with previous versions.

MINOR is incremented when new functionality is added in a manner that is backward-compatible with previous versions.

PATCH is incremented when bug fixes are made in a backward-compatible manner.

Two distinct leaves in the YANG module contains this semver semantic:

the semantic-version leaf contains the value reported as metadata by a specific YANG module.

the derived-semantic-version leaf is established by examining the the YANG module themselves. As such, only the YANG syntax, as opposed to the implementation changes that lead some some semantic changes.

Typically, an Openconfig YANG module would contain an extension, which is mapped to the semantic-version leaf.

```
   // extension statements
   extension openconfig-version {
     argument "semver" {
       yin-element false;
     }
     description
       "The OpenConfig version number for the module. This is
       expressed as a semantic version number of the form:
         x.y.z
        where:
         * x corresponds to the major version,
         * y corresponds to a minor version,
         * z corresponds to a patch version.
       This version corresponds to the model file within which it is
       defined, and does not cover the whole set of OpenConfig models.
       Where several modules are used to build up a single block of
       functionality, the same module version is specified across each
       file that makes up the module.

       A major version number of 0 indicates that this model is still
       in development (whether within OpenConfig or with industry
       partners), and is potentially subject to change.

       Following a release of major version 1, all modules will
       increment major revision number where backwards incompatible
       changes to the model are made.

       The minor version is changed when features are added to the
       model that do not impact current clients use of the model.

       The patch-level version is incremented when non-feature changes
       (such as bugfixes or clarifications to human-readable
       descriptions that do not impact model functionality) are made
       that maintain backwards compatibility.

       The version number is stored in the module meta-data.";
   }
```

Note that the absolute numbers in the semantic-version and derived-semantic-version are actually meaningless: the difference between two YANG module semver fields should be looked at.

In addition to the semantic versions, the yang-tree field points to the respective module’s simplified graphical representation of its model as described by [I-D.ietf-netmod-yang-tree-diagrams]. This diagram can be compared between two revisions of the same module to visually determine any structural differences when MAJOR or MINOR semantic versions differ.

4. YANG Catalog YANG module

The structure of the model defined in this document is described by the YANG module below.

```
<CODE BEGINS> file "yang-catalog@2018-04-03.yang"
  module yang-catalog {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:yang-catalog";
    prefix yc;

    import ietf-yang-types {
      prefix yang;
    }
    import ietf-yang-library {
      prefix yanglib;
    }
    import ietf-inet-types {
      prefix inet;
    }

    organization
      "yangcatalog.org";
    contact
      "Benoit Claise <bclaise@cisco.com>

       Joe Clarke <jclarke@cisco.com>";
    description
      "This module contains metadata pertinent to each YANG module, as
       well as a list of vendor implementations for each module.  The
       structure is laid out in such a way as to make it possible to
       locate metadata and vendor implementation on a per-module basis
       as well as obtain a list of available modules for a given
       vendor’s platform and specific software release.";

    revision 2018-04-03 {
      description
        "Bump the YANG version number to 1.1 for the deref XPath
         function.";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2018-01-23 {
      description
        "* Add leafs to track expire modules
         * Correct a bug with leafref dereferencing";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-09-26 {
```

```
      description
        "* Add leafs for tracking dependencies and dependents
         * Simplify the generated-from enumerated values
         * Refine the type for compilation-result to be an inet:uri
         * Add leafs for semantic versioning";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-08-18 {
      description
        "* Reorder organization to be with the other module keys
         * Add a belongs-to leaf to track a submodule's parent";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-28 {
      description
        "* Revert config false nodes as we need to be able to set these via <edi
t-config>

         * Make conformance-type optional as not all vendors implement yang-libr
ary

         * Re-add the path typedef";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-26 {
      description
        "A number of improvements based on YANG Doctor review:

         * Remove references to 'server' in leafs describing YANG data
         * Fold the augmentation module leafs directly under /catalog/modules/mo
dule
         * Use identities for protocols instead of an emumeration
         * Make some extractable fields 'config false'
         * Fix various types
         * Normalize enums to be lowercase
         * Add a leaf for module-classification
         * Change yang-version to be an enum
         * Add module conformance, deviation and feature leafs under the impleme
ntation branches";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-14 {
      description
        "Modularize some of the leafs and create typedefs so they
         can be shared between the API input modules.";
      reference "YANG Catalog <https://yangcatalog.org>";
    }
    revision 2017-07-03 {
      description
        "Initial revision.";
      reference
        "
```

```
         YANG Catalog <https://yangcatalog.org>";
    }

    /*
     * Identities
     */

    identity protocol {
      description
        "Abstract base identity for a YANG-based protocol.";
    }

    identity netconf {
      base protocol;
      description
        "Protocol identity for NETCONF as described in RFC 6241.";
    }

    identity restconf {
      base protocol;
      description
        "Protocol identity for RESTCONF as described in RFC 8040.";
    }

    typedef email-address {
      type string {
        pattern "[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+([.][a-zA-Z0-9-]+
)*";
      }
      description
        "This type represents a string with an email address.";
    }

    /*
     * Typedefs
     */

    typedef path {
      type string {
        pattern '([A-Za-z]:|[\w-]+(\.[\w-]+)*)?(([/\\][\w@.-]+)+)';
      }
      description
        "This type represents a string with path to the file.";
    }

    typedef semver {
      type string {
        pattern '[0-9]+\.[0-9]+\.[0-9]+';
      }
```

```
      description
        "A semantic version in the format of x.y.z, where:

         x = the major version number
         y = the minor version number
         z = the patch version number

         Changes to the major version number denote backwards-incompatible
         changes between two revisions of the same module.

         Changes to the minor version number indicate there have been new
         backwards-compatible features introduced in the later version of
         a module.

         Changes to the patch version indicate bug fixes between two
         versions of a module.";
      reference "Semantic Versioning 2.0.0 <http://semver.org/>";
    }

    container catalog {
      description
        "Root container of yang-catalog holding two main branches -
         modules and vendors. The modules sub-tree contains all the modules in
         the catalog and all of their metadata with their implementations.
         The vendor sub-tree holds modules for specific vendors, platforms,
         software-versions, and software-flavors. It contains reference to a
         name and revision of the module in order to reference the module’s full
         set of metadata.";
      container modules {
        description
          "Container holding the list of modules";
        list module {
          key "name revision organization";
          description
            "Each entry represents one revision of one module
             for one organization.";
          uses yang-lib-common-leafs;
          leaf organization {
            type string;
            description
              "This statement defines the party responsible for this
               module.  The argument is a string that is used to specify a textu
al
               description of the organization(s) under whose auspices this modu
le
               was developed.";
          }
          uses organization-specific-metadata;
          leaf namespace {
            type inet:uri;
```

```
            mandatory true;
            description
              "The XML namespace identifier for this module.";
          }
          uses yang-lib-schema-leaf;
          uses catalog-module-metadata;
          list submodule {
            key "name revision";
            description
              "Each entry represents one submodule within the
               parent module.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          list dependencies {
            key "name";
            description
              "Each entry represents one dependency.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          list dependents {
            key "name";
            description
              "Each entry represents one dependent.";
            uses yang-lib-common-leafs;
            uses yang-lib-schema-leaf;
          }
          leaf semantic-version {
            type yc:semver;
            description
              "The formal semantic version of a module as provided by the module
               itself.  If the module does not provide a semantic version, this
leaf

               will not be specified.";
          }
          leaf derived-semantic-version {
            type yc:semver;
            description
              "The semantic version of a module as compared to other revisions o
f

               the same module.  This value is computed algorithmically by order
ing

               all revisions of a given module and comparing them to look for ba
ckwards

               incompatible changes.";
          }
          container implementations {
            description
              "Container holding lists of per-module implementation details.";
            list implementation {
              key "vendor platform software-version software-flavor";
```

```
              description
                "List of module implementations.";
              leaf vendor {
                type string;
                description
                  "Organization that implements this module.";
              }
              leaf platform {
                type string;
                description
                  "Platform on which this module is implemented.";
              }
              leaf software-version {
                type string;
                description
                  "Name of the version of software.  With respect to most networ
k device appliances,
                   this will be the operating system version.  But for other YAN
G module
                   implementation, this would be a version of appliance software
.  Ultimately,
                   this should correspond to a version string that will be recog
nizable by
                   the consumers of the platform.";
              }
              leaf software-flavor {
                type string;
                description
                  "A variation of a specific version where
                   YANG model support may be different.  Depending on the vendor
, this could
                   be a license, additional software component, or a feature set
.";
              }
              uses shared-implementation-leafs;
              uses yang-lib-implementation-leafs;
            }
          }
        }
      }
      container vendors {
        description
          "Container holding lists of organizations that publish YANG modules.";
        list vendor {
          key "name";
          description
            "List of organizations publishing YANG modules.";
          leaf name {
            type string;
            description
              "Name of the maintaining organization -- the name should be
               supplied in the official format used by the organization.
               Standards Body examples:
                 IETF, IEEE, MEF, ONF, etc.
```

```
               Commercial entity examples:
                 AT&T, Facebook, <Vendor>
               Name of industry forum examples:
               OpenConfig, OpenDaylight, ON.Lab";
          }
          container platforms {
            description
              "Container holding list of platforms.";
            list platform {
              key "name";
              description
                "List of platforms under specific vendor";
              leaf name {
                type string;
                description
                  "Name of the platform";
              }
              container software-versions {
                description
                  "Container holding list of versions of software versions.";
                list software-version {
                  key "name";
                  description
                    "List of version of software versions under specific vendor,
 platform.";
                  leaf name {
                    type string;
                    description
                      "Name of the version of software.  With respect to most ne
twork device appliances,
                       this will be the operating system version.  But for other
 YANG module
                       implementation, this would be a version of appliance soft
ware.  Ultimately,
                       this should correspond to a version string that will be r
ecognizable by
                       the consumers of the platform.";
                  }
                  container software-flavors {
                    description
                      "Container holding list of software flavors.";
                    list software-flavor {
                      key "name";
                      description
                        "List of software flavors under specific vendor, platfor
m, software-version.";
                      leaf name {
                        type string;
                        description
                          "A variation of a specific version where
                           YANG model support may be different.  Depending on th
e vendor, this could
                           be a license, additional software component, or a fea
ture set.";
                      }
                      container protocols {
```

```
                        description
                          "List of the protocols";
                        list protocol {
                          key "name";
                          description
                            "YANG-based protocol that is used on the device.  Ne
w identities
                             are expected to be added to address other YANG-base
d protocols.";
                          leaf name {
                            type identityref {
                              base yc:protocol;
                            }
                            description
                              "Identity of the YANG-based protocol that is suppo
rted.";
                          }
                          leaf-list protocol-version {
                            type string;
                            description
                              "Version of the specific protocol.";
                          }
                          leaf-list capabilities {
                            type string;
                            description
                              "Listed name of capabilities that are
                               supported by the specific device.";
                          }
                        }
                      }
                      container modules {
                        description
                          "Container holding list of modules.";
                        list module {
                          key "name revision organization";
                          description
                            "List of references to YANG modules under specific v
endor, platform, software-version,
                             software-flavor.  Using these references, the compl
ete set of metadata can be
                             retrieved for each module.";
                          leaf name {
                            type leafref {
                              path "/catalog/modules/module/name";
                            }
                            description
                              "Reference to a name of the module that is contain
ed in specific vendor, platform,
                               software-version, software-flavor.";
                          }
                          leaf revision {
                            type leafref {
                              path "deref(../name)/../revision";
                            }
```

```
                            description
                              "Reference to a revision of the module that is con
tained in specific vendor,
                               platform, software-version, software-flavor.";
                          }
                          leaf organization {
                            type leafref {
                              path "deref(../revision)/../organization";
                            }
                            description
                              "Reference to the authoring organization of the mo
dule for the implemented
                                module.";
                          }
                          uses shared-implementation-leafs;
                          uses yang-lib-implementation-leafs;
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }

    grouping catalog-module-metadata {
      uses shared-module-leafs;
      leaf compilation-status {
        type enumeration {
          enum passed {
            description
              "All compilers were able to compile this YANG module without
               any errors or warnings.";
          }
          enum passed-with-warnings {
            description
              "All compilers were able to compile this YANG module without
               any errors, but at least one of them caught a warning.";
          }
          enum failed {
            description
              "At least one of compilers found an error while
               compiling this YANG module.";
          }
          enum pending {
            description
              "The module was just added to the catalog and compilation testing
is still
```

```
               in progress.";
          }
          enum unknown {
            description
              "There is not sufficient information about compilation status.  Th
is Could
               mean compilation crashed causing it not to complete fully.";
          }
        }
        description
          "Status of the module, whether it was possible to compile this YANG mo
dule or
           there are still some errors/warnings.";
      }
      leaf compilation-result {
        type inet:uri;
        description
          "Link to the result of the compilation explaining specifically what er
ror or
           warning occurred.  This is not existing if compilation status is PASS
ED.";
      }
      leaf prefix {
        type string;
        description
          "Statement of yang that is used to define the prefix associated with
           the module and its namespace. The prefix statement's argument is
           the prefix string that is used as a prefix to access a module. The
           prefix string MAY be used to refer to definitions contained in the
           module, e.g., if:ifName.";
      }
      leaf yang-version {
        type enumeration {
          enum 1.0 {
            description
              "YANG version 1.0 as defined in RFC 6020.";
          }
          enum 1.1 {
            description
              "YANG version 1.1 as defined in RFC 7950.";
          }
        }
        description
          "The optional yang-version statement specifies which version of the
           YANG language was used in developing the module.";
      }
      leaf description {
        type string;
        description
          "This statement takes as an argument a string that
           contains a human-readable textual description of this definition.
           The text is provided in a language (or languages) chosen by the
```

```
           module developer; for the sake of interoperability, it is RECOMMENDED
           to choose a language that is widely understood among the community of
           network administrators who will use the module.";
      }
      leaf contact {
        type string;
        description
          "This statement provides contact information for the module.
           The argument is a string that is used to specify contact information
           for the person or persons to whom technical queries concerning this
           module should be sent, such as their name, postal address, telephone
           number, and electronic mail address.";
      }
      leaf module-type {
        type enumeration {
          enum module {
            description
              "If YANG file contains module.";
          }
          enum submodule {
            description
              "If YANG file contains sub-module.";
          }
        }
        description
          "Whether a file contains a YANG module or sub-module.";
      }
      leaf belongs-to {
        when "../module-type = 'submodule'" {
          description
            "Include the module's parent when it is a submodule.";
        }
        type yang:yang-identifier;
        description
          "Name of the module that includes this submodule.";
      }
      leaf tree-type {
        type enumeration {
          enum split {
            description
              "This module uses a split config/operational state layout.";
          }
          enum nmda-compatible {
            description
              "This module is compatible with the Network Management Datastores
               Architecture (NMDA) and combines config and operational state nod
es.";
          }
          enum transitional-extra {
```

```
            description
              "This module is derived as a '-state' module to allow for transiti
oning
               to a full NMDA-compliant tree structure.";
          }
          enum openconfig {
            description
              "This module uses the Openconfig data element layout.";
          }
          enum unclassified {
            description
              "This module does not belong to any category or can't be determine
d.";
          }
          enum not-applicable {
            description
              "This module is not applicable. For example, because the YANG modu
le only contains typedefs, groupings, or is a submodule";
          }
        }
        description
          "The type of data element tree used by the module as it relates to the
           Network Management Datastores Architecture.";
        reference "draft-dsdt-nmda-guidelines Guidelines for YANG Module Authors
 (NMDA)";
      }
      leaf yang-tree {
        when "../module-type = 'module'";
        type inet:uri;
        description
          "This leaf provides a URI that points to the ASCII tree format of the
module in
           draft-ietf-netmod-yang-tree-diagrams format.";
        reference "See draft-ietf-netmod-yang-tree-diagrams.";
      }
      leaf expires {
        type yang:date-and-time;
        description
          "Date and time of when this module expires (if it expires).  This will
 typically be used for
           modules that have not been fully ratified.";
      }
      leaf expired {
        type union {
          type boolean;
          type enumeration {
            enum not-applicable {
              description
                "This module is not and will not be expired.";
            }
          }
        }
        default "false";
        description
```

```
          "Whether or not this module has expired.  If the current date is beyon
d the expires date, then expired
           should be true.";
      }
      description
        "Grouping of YANG module metadata that extends the common list defined i
n the YANG
         Module Library (RFC 7895).";
    }

    grouping organization-specific-metadata {
      container ietf {
        when "../organization = 'ietf'" {
          description
            "Include this container specific metadata of the IETF.";
        }
        leaf ietf-wg {
          type string;
          description
            "Working group that authored the document containing this module.";
        }
        description
          "Include this container for the IETF-specific organization metadata.";
      }
      description
        "Any organization that has some specific metadata of the yang module and
 want them add to the
         yang-catalog, should augment this grouping. This grouping is for any me
tadata that can't be used for
         every yang module.";
    }

    grouping yang-lib-common-leafs {
      leaf name {
        type yang:yang-identifier;
        description
          "The YANG module or submodule name.";
      }
      leaf revision {
        type union {
          type yanglib:revision-identifier;
          type string {
            length "0";
          }
        }
        description
          "The YANG module or submodule revision date.
           A zero-length string is used if no revision statement
           is present in the YANG module or submodule.";
      }
      description
        "The YANG module or submodule revision date.
```

```
         A zero-length string is used if no revision statement
         is present in the YANG module or submodule.";
      reference "RFC7895 YANG Module Library : common-leafs grouping";
    }

    grouping yang-lib-schema-leaf {
      leaf schema {
        type inet:uri;
        description
          "Contains a URL that represents the YANG schema
           resource for this module or submodule.
           This leaf will only be present if there is a URL
           available for retrieval of the schema for this entry.";
      }
      description
        "These are a subset of leafs from the yang-library (RFC 7895) that provi
de some
         extractable fields for catalog modules.  The module-list grouping canno
t be
         used from yang-library as modules themselves cannot have conformance wi
thout
         a server.";
      reference "RFC7895 YANG Module Library : schema-leaf grouping";
    }

    grouping yang-lib-implementation-leafs {
      leaf-list feature {
        type yang:yang-identifier;
        description
          "List of YANG feature names from this module that are
           supported by the server, regardless of whether they are
           defined in the module or any included submodule.";
      }
      list deviation {
        key "name revision";
        description
          "List of YANG deviation module names and revisions
           used by this server to modify the conformance of
           the module associated with this entry.  Note that
           the same module can be used for deviations for
           multiple modules, so the same entry MAY appear
           within multiple 'module' entries.
           The deviation module MUST be present in the 'module'
           list, with the same name and revision values.
           The 'conformance-type' value will be 'implement' for
           the deviation module.";
        uses yang-lib-common-leafs;
      }
      leaf conformance-type {
        type enumeration {
          enum implement {
```

```
            description
              "Indicates that the server implements one or more
               protocol-accessible objects defined in the YANG module
               identified in this entry.  This includes deviation
               statements defined in the module.
               For YANG version 1.1 modules, there is at most one
               module entry with conformance type 'implement' for a
               particular module name, since YANG 1.1 requires that,
               at most, one revision of a module is implemented.
               For YANG version 1 modules, there SHOULD NOT be more
               than one module entry for a particular module name.";
          }
          enum import {
            description
              "Indicates that the server imports reusable definitions
               from the specified revision of the module but does
               not implement any protocol-accessible objects from
               this revision.
               Multiple module entries for the same module name MAY
               exist.  This can occur if multiple modules import the
               same module but specify different revision dates in
               the import statements.";
          }
        }
        // Removing the mandatory true for now as not all vendors may have
        // this information if they do not implement yang-library.
        //mandatory true;
        description
          "Indicates the type of conformance the server is claiming
           for the YANG module identified by this entry.";
      }
      description
        "This is a set of leafs extracted from the yang-library that are
         specific to server implementations.";
      reference "RFC7895 YANG Module Library : module-list grouping";
    }

    grouping shared-implementation-leafs {
      leaf os-version {
        type string;
        description
          "Version of the operating system using this module.  This is primarily
 useful if
           the software implementing the module is an application that requires
a specific
           operating system.";
      }
      leaf feature-set {
        type string;
        description
```

```
          "An optional feature of the software that is required in order to impl
ement this
           module.  Some form of this must be incorporated in software-version o
r
           software-flavor, but can be broken out here for additional clarity.";
      }
      leaf os-type {
        type string;
        description
          "Type of the operating system using this module.  This is primarily us
eful if
           the software implementing the module is an application that requires
a
           specific operating system.";
      }
      description
        "Grouping of non-key leafs to be used in the module and vendor sub-trees
.";
    }

    grouping shared-module-leafs {
      leaf generated-from {
        type enumeration {
          enum mib {
            description
              "Module generated from Structure of Management Information (SMI)
               MIB per RFC6643.";
          }
          enum not-applicable {
            description
              "Module was not generated but it was authored manually.";
          }
          enum native {
            description
              "Module generated from platform internal,
               proprietary structure, or code.";
          }
        }
        default "not-applicable";
        description
          "This statement defines weather the module was generated or not.
           Default value is set to not-applicable, which means that module
           was created manualy and not generated.";
      }
      leaf maturity-level {
        type enumeration {
          enum ratified {
            description
              "Maturity of a module that is fully approved (e.g., a standard).";
          }
          enum adopted {
            description
              "Maturity of a module that is actively being developed by a organi
zation towards ratification.";
```

```
          }
          enum initial {
            description
              "Maturity of a module that has been initially created, but has no
official
               organization-level status.";
          }
          enum not-applicable {
            description
              "The maturity level is not used for vendor-supplied models, and th
us all vendor
               modules will have a maturity of not-applicable";
          }
        }
        description
          "The current maturity of the module with respect to the body that crea
ted it.
           This allows one to understand where the module is in its overall life
 cycle.";
      }
      leaf document-name {
        type string;
        description
          "The name of the document from which the module was extracted or taken
;
           or that provides additional context about the module.";
      }
      leaf author-email {
        type yc:email-address;
        description
          "Contact email of the author who is responsible for this module.";
      }
      leaf reference {
        type inet:uri;
        description
          "A string that is used to specify a textual cross-reference to an exte
rnal document, either
           another module that defines related management information, or a docu
ment that provides
           additional information relevant to this definition.";
      }
      leaf module-classification {
        type enumeration {
          enum network-service {
            description
              "Network Service YANG Module that describes the configuration, sta
te
               data, operations, and notifications of abstract representations o
f
               services implemented on one or multiple network elements.";
          }
          enum network-element {
            description
              "Network Element YANG Module that describes the configuration, sta
te
               data, operations, and notifications of specific device-centric
               technologies or features.";
          }
```

```
          enum unknown {
            description
              "In case that there is not sufficient information about how to cla
ssify the module.";
          }
          enum not-applicable {
            description
              "The YANG module abstraction type is neither a Network Service YAN
G Module
               nor a Network Element YANG Module.";
          }
        }
        mandatory true;
        description
          "The high-level classification of the given YANG module.";
        reference "RFC8199 YANG Module Classification";
      }
      description
        "These leafs are shared among the yang-catalog and its API.";
    }

    grouping online-source-file {
      leaf owner {
        type string;
        mandatory true;
        description
          "Username or ID of the owner of the version control system repository.
";
      }
      leaf repository {
        type string;
        mandatory true;
        description
          "The name of the repository.";
      }
      leaf path {
        type yc:path;
        mandatory true;
        description
          "Location within the repository of the module file.";
      }
      leaf branch {
        type string;
        description
          "Revision control system branch or tag to use to find the module.  If
this is not
           specified, the head of the repository is used.";
      }
      description
        "Networked version control system location of the module file.";
    }
  }
```

<CODE ENDS>

5. Security Considerations

The goal of the YANG Catalog module and yangcatalog.org is to document a large library of YANG modules and their implementations. Already, we have seen some SDOs hestitant to provide modules that have not reached a "ratified" maturity level because of intellectual property leakage concerns or simply organization process that mandates only fully ratified modules can be published. Care must be paid that through private automated testing and validation of such modules that their metadata does not leak before the publishing organization approves the release of such data.

Similarly, from a vendor implementation standpoint, data that is exposed to the catalog before the vendor has fully vetted it could cause confusion amongst that vendor’s customers or reveal product releases to the market before they have been officially announced.

Ultimately, there is a balance to be struck with respect to providing a rich library of YANG module metadata, and doing so at the right time to avoid information leakage.

6. IANA Considerations

No IANA action is requested.

7. References

7.1. Normative References

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

[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>.

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

[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>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

[semver] "Semantic Versioning 2.0.0", <https://www.semver.org>.

7.2. Informative References

[I-D.openconfig-netmod-model-catalog]
Shaikh, A., Shakir, R., and K. D'Souza, "Catalog and registry for YANG models", draft-openconfig-netmod-model-catalog-02 (work in progress), March 2017.

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

7.3. URIs

[1] https://developer.cisco.com/site/confD/index.gsp

[2] https://www.yangcatalog.org/yang-search

Appendix A. Acknowledgments

The authors would like to thanks Miroslav Kovac for this help on this YANG module and the yangcatalog.org implementation. We would also like to thank Radek Krejci for his extensive review and suggestions for improvement.

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

Appendix B. Changes From Previous Revisions

RFC Editor to remove this section prior to publication.

Draft -00 to -01:

- o Redesign of module sub-tree based on review.
- o Modularize some leafs and create typedefs to share with API YANG modules.
- o Add module conformance-type, deviation and feature leafs under the implementation branch.
- o Change yang-version to be an enum.
- o Add a leaf for module-classification based on [RFC8199].
- o Normalize enums to be lowercase.
- o Use identities for protocols instead of an enumeration.
- o Make conformance-type optional as not all vendors implement [RFC7895].
- o Add a leaf for tree-type based on [RFC8342].
- o Add a reference to contributing to the YANG Catalog at yangcatalog.org.
- o Various wording and style changes to the document text.

Draft -01 to -02:

- o Add a belongs-to leaf to track parent modules.
- o Add leafs to track dependents and dependencies for a given module.
- o Simplify the generated-from enumerated values.
- o Refine the type for compilation-result to be an inet:uri.
- o Add leafs for semantic versioning.
- o Reorder the organization leaf to be with other module keys.
- o Add text to describe generated-from and semantic versioning.

Draft -02 to -03:

- o Change YANG ref to RFC7950 as the catalog module now needs YANG 1.1.
- o Add a reference to I-D.ietf-netmod-yang-tree-diagrams.
- o Document the new yang-tree node in the catalog.
- o Document the new expires and expired leafs and their relation to maturity.
- o Updtae NMDA reference to point to new RFC number.

Authors' Addresses

Joe Clarke
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: January 3, 2019

B. Claise
J. Clarke
Cisco Systems, Inc.
B. Lengyel
Ericsson
K. D'Souza
AT&T

July 2, 2018

New YANG Module Update Procedure
draft-clacla-netmod-yang-model-update-06

Abstract

This document specifies a new YANG module update procedure in case of backward-incompatible changes, as an alternative proposal to the YANG 1.1 specifications. This document updates RFC 7950.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document puts forth a solution to the problems described in [I-D.verdt-netmod-yang-versioning-reqs] by proposing changes to [RFC7950] to address the various requirements that [I-D.verdt-netmod-yang-versioning-reqs] specifies. At this time, the solution herein addresses requirements 1.1, 1.2, 1.3, 2.1, 4.1, 4.2, 4.3, 5.1, and 5.2. Current gaps are documented in Appendix A.1 below.

2. The Solution

The solution is composed of five parts:

1. A semantic versioning YANG extension, along with an optional additional check that validates the semantic versioning from a syntactic point of view, which can either assist in determining the correct semantic versioning value, or which can help in

determining the values for YANG modules that do not support this extension.

2. An import by semantic version statement

3. Updates to the YANG 1.1 module update rules

4. Updates to ietf-yang-library

5. An introduction of deprecated and obsolote reason clauses

2.1. Semantic Versioning

2.1.1. Semantic Versioning, As Set by the YANG Module Designer

The semantic versioning solution proposed here has already been proposed in [I-D.openconfig-netmod-model-catalog] (included here with the authors' permission) which itself is based on [openconfigsemver]. The goal is to indicate the YANG module backward (in)compatibility, following semver.org semantic versioning [semver]:

"The SEMVER version number for the module is introduced. This is expressed as a semantic version number of the form: x.y.z

- x is the MAJOR version. It is incremented when the new version of the specification is incompatible with previous versions.

- y is the MINOR version. It is incremented when new functionality is added in a manner that is backward-compatible with previous versions.

- z is the PATCH version. It is incremented when bug fixes are made in a backward-compatible manner."

The semantic version value is set by the YANG module developer at the design and implementation times. Along these lines, we propose the following YANG 1.1 extension for a more generic semantic version. The formal definition is found at the end of this document. This semantic version extension and the text below address requirements 1.1, 1.2, 2.1, 5.1 and 5.2 of [I-D.verdt-netmod-yang-versioning-reqs].

```
extension module-version {
    argument semver;
}
```

The extension would typically be used this way:

```
module yang-module-name {

  namespace "name-space";
  prefix "prefix-name";

  import ietf-semver { prefix "semver"; }

  description
    "to be completed";


  revision 2017-10-30 {
    description
      "Change the module structure";
    semver:module-version "2.0.0";
  }

  revision 2017-07-30 {
    description
      "Added new feature XXX";
    semver:module-version "1.2.0";
  }

  revision 2017-04-03 {
    description
      "Update copyright notice.";
    semver:module-version "1.0.1";
  }

  revision 2017-04-03 {
    description
      "First release version.";
    semver:module-version "1.0.0";
  }

  revision 2017-01-26 {
    description
      "Initial module for inet types";
    semver:module-version "0.1.0";
  }


  //YANG module definition starts here
```

See also "Semantic Versioning and Structure for IETF Specifications" [I-D.claise-semver] for a mechanism to combine the semantic versioning, the GitHub tools, and a potential change to the IETF process.

2.1.2. The Derived Semantic Version

If an explicitly defined semantic version is not available in the YANG module, it is possible to algoritmically calculate a derived semantic version. This can be used for modules not containing a definitive semantic-version as defined in this document or as a starting value when specifying the definitive semantic-version. Be aware that this algorithm may sometimes incorrectly classify changes between the categories non-compatible, compatible or error-correction.

2.1.3. Implementation Experience

[yangcatalog] uses the pyang utility to calculate the derived-semantic-version for all of the modules contained within the catalog. [yangcatalog] contains many revisions of the same module in order to provide its derived-semantic-version for module consumers to know what has changed between revisions of the same module.

Two distinct leafs in the YANG module [I-D.clacla-netmod-model-catalog] contain this semver notation:

- the semantic-version leaf contains the value embedded within a YANG module (if it is available).

- the derived-semantic-version leaf is established by examining the the YANG module themselves. As such derived-semantic-version only takes syntax into account as opposed to the meaning of various elements when it computes the semantic version.

- The algorithm used to produce the derived-semantic-version is as follows:

  1. Order all modules of the same name by revision from oldest to newest. Include module revisions that are not available, but which are defined in the revision statements in one of the available module versions.

  2. If module A, revision N+1 has failed compilation, bump its derived semantic MAJOR version. For unavailable module versions assume non-backward compatible changes were done., thus bump its derived semantic MAJOR version.

  3. Else, run "pyang --check-update-from" on module A, revision N and revision N+1 to see if backward-incompatible changes exist.

4. If backward-incompatible changes exist, bump module A, revision N+1’s derived MAJOR semantic version.

5. If no backward-incompatible changes exist, compare the pyang trees of module A, revision N and revision N+1.

6. If there are structural differences (e.g., new nodes), bump module A, revision N+1’s derived MINOR semantic version.

7. If no structural differences exist, bump module A, revision N+1’s derived PATCH semantic version.

The pyang utility checks many of the points listed in section 11 of [RFC7950] for known module incompatibilities. While this approach is a good way to programmatically obtain a semantic version number, it does not address all cases whereby a major version number might need to be increased. For example, a node may have the same name and same type, but its meaning may change from one revision of a module to another. This represents a semantic change that breaks backward compatibility, but the above algorithm would not find it. Therefore, additional, sometimes manual, rigor must be done to ensure a proper version is chosen for a given module revision.

2.2. Import by Semantic Version

If a module is imported by another one, it is usually not specified which revision of the imported module should be used. However, not all revisions may be acceptable. Today YANG 1.1 allows one to specify the revision date of the imported module, but that is too specific, as even a small spelling correction of the imported module results in a change to its revision date, thus making the module revision ineligible for import.

Using semantic versioning to indicate the acceptable imported module versions is much more flexible. For example:

- Only a module of a specific MAJOR version is acceptable. All MINOR and PATCH versions can also be imported.

- A module at a specific MAJOR version or higher is acceptable.

- A module at a specific MAJOR.MINOR version is acceptable. All PATCH versions can also be imported.

- A module within a certain range of versions are acceptable. For example, in this case, a module between version 1.0.0 (inclusive) and 3.0.0 (exclusive) are acceptable.

The ietf-semver module provides another extension, import-versions that is a child of import and specifies the rules for an acceptable set of versions of the given module. This extension addresses requirement 1.3 of [I-D.verdt-netmod-yang-versioning-reqs]. The structure of this extension is specified as follows:

TODO: How to specify this? One thought is below, not fully formalized as this should be discussed further. Note: while this uses a comma to separate discrete versions, we could instead allow for this to be specified multiple times.

[\[(]X[.Y[.Z]][-[X[.Y[.X]]][\])]][,...]

Where the first character MAY be a '[' or '(' to indicate at least inclusive and at least exclusive (respectively). If this is omitted, a full semantic version must be specified and the import will only support this one version.

The following version, if specified with a '[' or '(' indicates the lower bound. This can be a full semantic version or a MAJOR only or MAJOR.MINOR only.

The '-', if specified, is a literal hyphen indicating a range will be specified. If the second portion of the import-versions clause is omitted, then there is no upper bound on what will be considered an acceptable imported version.

After the '-' the upper bound semantic version (or part thereof) follows.

After the upper bound version, one of ']' or ')' MUST follow to indicate whether this limit is inclusive or exclusive of the upper bound respectively.

Finally, a literal comma (',') MAY be specified with additional ranges. Each range is taken as a logical OR.

For example:

```
import example-module {
  semver:import-versions "[1.0.0-3.0.0)";
  // All versions between 1.0.0 (inclusive) and 3.0.0 (exclusive) are acceptable
.
}

import example-module {
  semver:import-versions "[2-5]";
  // All versions between 2.0.0 (inclusive) and 5.y.z (inclusive) where y and z
are
  // any value for MINOR and PATCH versions.
}

import example-module {
  semver:import-versions "[1.5-2.0.0),[2.5";
  // All versions between 1.5.0 (inclusive) and 2.0.0 (exclusive) as well as all
 versions
  // greater than 2.5 (inclusive).  In this manner, if 2.0 was branched from 1.4
, and a
  // new feature was added into 1.5, all versions of 1.x.x starting at 1.5 are a
llowed,
  // but the feature was not merged into 2.y.z until 2.5.0.
}

import example-module {
  semver:import-versions "[1";
  // All versions greater than MAJOR version 1 are acceptable.  This includes an
y
  // MINOR or PATCH versions.
}

import example-module {
  semver:import-versions "1.0.0";
  // Only version 1.0.0 is acceptable (this mimics what exists with import by re
vision).
}

import example-module {
  semver:import-versions "[1.1-2)"";
  // All versions greater than 1.1 (inclusive, and including all PATCH versions
off of 1.1)
  // up to MAJOR version 2 (exclusive) are acceptable.
}

import example-module {
  semver:import-versions "[1.1-2),[3";
  // All versions greater than 1.1 (inclusive, and including all PATCH versions
off of 1.1)
  // up to MAJOR version 2 (exclusive), as well as all versions greater than MAJ
OR version 3
  // (inclusive) are acceptable.
}

import example-module {
  semver:import-versions "[1.1-2),[3.0.0";
  // This is equivalent to the example above, simply indicating that a partial s
emantic version
  // assumes all missing components are 0.
}
```

The import statement SHOULD include a semver:import-versions statement and MUST NOT include a revision statement. An import statement MUST NOT contain both a semver:import-versions and a revision substatement. The use of the revision substatement for import should be discouraged.

2.3. Updates to YANG 1.1 Module Update Rules

RFC 7950 section 11, must be updated to allow for non-backward changes provided they follow the semantic versioning guidelines and increase the MAJOR version number when a backward incompatible change is made. This change is in the spirit of requirement 5.1 from [I-D.verdt-netmod-yang-versioning-reqs]. The following is proposed text for this change.

"As experience is gained with a module, it may be desirable to revise that module. Changes to published modules are allowed, even if they have some potential to cause interoperability problems, if the module-version YANG extension is used in the revision statement to clearly indicate the nature of the change."

2.4. Updates to ietf-yang-library

The ietf-semver YANG module also specifies additional ietf-yang-library [RFC7895] [I-D.ietf-netconf-rfc7895bis] leafs to be added at the module and submodule levels. The first is module-version, which augments /yanglib:yang-library/yanglib:module-set/yanglib:module. This specifies the current semantic version of the associated module and revision in a given module-set. The related submodule-version leaf is added at /yanglib:yang-library/yanglib:module-set/yanglib:module/yanglib:submodule to indicate the semantic version of a submodule.

In order to satisfy the requirements 4.1 and 4.3 of [I-D.verdt-netmod-yang-versioning-reqs] that deprecated and obsolete node presence and operation are easily and clearly known to clients, ietf-semver also augments the ietf-yang-library with two additional boolean leafs at /yanglib:yang-library/yanglib:module-set/ yanglib:module. A client can make one request of the ietf-yang-library and know whether or a not a module that has deprecated or obsolete has those nodes implemented by the server, as opposed to making multiple requests for each node in question.

deprecated-nodes-present : A boolean that indicates whether or not this server implements deprecated nodes. The value of this leaf SHOULD be true; and if so, the server MUST implement nodes within this module as they are documented. If specific deprecated nodes are not implemented as documented, then they MUST be listed as deviations. This leaf defaults to true.

obsolete-nodes-present : A boolean that indicates whether or not this server implements obsolete nodes. The value of this leaf SHOULD be false; and if so, the server MUST NOT implement nodes within this module. If this leaf is true, then all nodes in this module MUST be implemented as documented in the module. Any variation of this MUST be listed as deviations. This leaf defaults to false.

If a module does not have any deprecated or obsolete nodes, the server SHOULD set the corresponding leaf above to true. This is helpful to clients, such that if the MAJOR version number has not changed, and these booleans are true, then a client does not have to check the status of any node for the module.

Module compatibility can be affected if values other than the default are used for the leafs described here. For example, if a server does not implemennt deprecated nodes, then a given module revision may be incompatible with a previous revision where the nodes were not deprecated. When calculating backward compatibility, the default values of these leafs MUST be considered. From a client's point of view, if two module revisions have the same MAJOR version but the run-time value of deprecated-nodes-present (as read from the ietf-yang-library) is false, then compatibility MUST NOT be assumed based on the module-version alone.

2.5. Deprecated and Obsolete Reasons

The ietf-semver module specifies an extension, status-description, that is designed to be used as a substatement of the status statement when the status is deprecated or obsolete. The argument to this extension is freeform text that explains why the node was deprecated or made obsolete. It may also point to other schema elements that take the place of the deprecated or obsolete node. This text is designed for human consumption to aid in the migration away from nodes that will one day no longer work. These extensions address requirement 4.2 of [I-D.verdt-netmod-yang-versioning-reqs]. An example is shown below.

```
   leaf imperial-temperature {
     type int64;
     units "degrees Fahrenheit";
     status deprecated {
       semver:status-description
         "Imperial measurements are being phased out in favor
          of their metric equivalents.  Use metric-temperature
          instead.";
     }
     description
       "Temperature in degrees Fahrenheit.";
   }
```

3. Semantic Version Extension YANG Module

The extension and related ietf-yang-library changes described in this module are defined in the YANG module below.

```
<CODE BEGINS> file "ietf-semver@2018-04-05.yang"
  module ietf-semver {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-semver";
    prefix semver;

    import ietf-yang-library {
      prefix yanglib;
    }

    organization
      "IETF NETMOD (Network Modeling) Working Group";
    contact
      "WG Web:   <https://datatracker.ietf.org/wg/netmod/>
       WG List:  <mailto:netmod@ietf.org>

       Author:   Benoit Claise
                 <mailto:bclaise@cisco.com>

       Author:   Joe Clarke
                 <mailto:jclarke@cisco.com>

       Author:   Kevin D'Souza
                 <mailto:kd6913@att.com>

       Author:   Balazs Lengyel
                 <mailto:balazs.lengyel@ericsson.com>";
    description
      "This module contains a definition for a YANG 1.1 extension to
       express the semantic version of YANG modules.";
```

```
     revision 2018-04-05 {
       description
         "* Properly import ietf-yang-library.
          * Fix the name of module-semver => module-version.
          * Fix regular expression syntax.
          * Augment yang-library with booleans as to whether or not
            deprecated and obsolete nodes are present.
          * Add an extension to enable import by semantic version.
          * Add an extension status-description to track deprecated
            and obsolete reasons.
          * Fix yang-library augments to use 7895bis.";
       reference
         "draft-clacla-netmod-yang-model-update:
          New YANG Module Update Procedure";
       semver:module-version "0.2.1";
     }
     revision 2017-12-15 {
       description
         "Initial revision.";
       reference
         "draft-clacla-netmod-yang-model-update:
          New YANG Module Update Procedure";
       semver:module-version "0.1.1";
     }

     extension module-version {
       argument semver;
       description
         "The version number for the module revision it is used in.
          This is expressed as a semantic version string in the form:
           x.y.z
          where:
           * x corresponds to the major version,
           * y corresponds to a minor version,
           * z corresponds to a patch version.

          A major version number of 0 indicates that this model is still
          in development, and is potentially subject to change.

          Following a release of major version 1, all modules will
          increment major revision number where backward incompatible
          changes to the model are made.

          The minor version is changed when features are added to the
          model that do not impact current clients use of the model.
          When major version is stepped, the minor version is reset to 0.

          The patch-level version is incremented when non-feature changes
```

```
          (such as bugfixes or clarifications to human-readable
          descriptions that do not impact model functionality) are made
          that maintain backward compatibility.
          When major or minor version is stepped, the patch-level is
          reset to 0.

          By comparing the module-version between two revisions of a
          given module, one can know if different revisions are backward
          compatible or not, as well as
          whether or not new features have been added to a newer revision.

          If a module contains this extension it indicates that for this
          module the updated status and update rules as this described in
          RFC XXXX are used.

          The statement MUST only be a substatement of the revision statement.
          Zero or one module-version statement is allowed per parent
          statement. NO substatements are allowed.
         ";
      reference "http://semver.org/ : Semantic Versioning 2.0.0";
    }

    extension import-versions {
      argument version-clause;
      description
        "This extension specifies an acceptable set of semantic versions of a gi
ven module
          that may be imported.  The version-clause argument is specified in the
following
          format

          [\\[(]X[.Y[.Z]][-[X[.Y[.X]]][\\])]][,...]

          Where the first character MAY be a '[' or '(' to indicate at least incl
usive and at least
           exclusive (respectively).  If this is omitted, a full semantic version
 must be specified
           and the import will only support this one version.

          The following version, if specified with a '[' or '(' indicates the low
er bound.  This can
           be a full semantic version or a MAJOR only or MAJOR.MINOR only.

          The '-', if specified, is a literal hyphen indicating a range will be s
pecified.  If the second portion
           of the import-versions clause is omitted, then there is no upper bound
 on what will be considered
           an acceptable imported version.

          After the '-' the upper bound semantic version (or part thereof) follow
s.
          After the upper bound version, one of ']' or ')' MUST follow to indicat
e whether this limit is inclusive
           or exclusive of the upper bound respectively.

          Finally, a literal comma (',') MAY be specified with additional ranges.
  Each range is taken as a logical
           OR.
```

```
         The statement MUST only be a substatement of the import statement.  Zer
o or one
         import-versions statement is allowed per import statement.  NO substate
ments are allowed.";
      reference "I-D.clacla-netmod-yang-model-update : Import By Semantic Versio
n";
    }

    extension status-description {
      argument description;
      description
        "Freeform text that describes why a given node has been deprecated or ma
de obsolete.
         This may point to other schema elements that can be used in lieu of the
 given node.

         This statement MUST only be used as a substatement of the status statem
ent, and MUST
         only be used when the status is deprecated or obsolete.  Zero or more s
tatus-description
         statements are allowed per parent statement.  NO substatements are allo
wed.";
      reference "I-D.clacla-netmod-yang-model-update : Deprecated and Obsolete R
easons";
    }

    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
      description
        "Augmentations for the ietf-yang-library module to support semantic vers
ioning.";
      leaf module-version {
        type string {
          pattern '[0-9]+\.[0-9]+\.[0-9]+';
        }
        description
          "The semantic version for this module in MAJOR.MINOR.PATCH format.  Th
is version
           must match the semver:module-version value in specific revision of th
e module
           loaded in this module-set.";
      }
      leaf deprecated-nodes-present {
        type boolean;
        default "true";
        description
          "A boolean that indicates whether or not this server implements deprec
ated nodes.
           The value of this leaf SHOULD be true; and if so, the server MUST imp
lement nodes
           within this module as they are documented.  If specific deprecated no
des are not
           implemented as document, then they MUST be listed as deviations.  If
a module does
           not currently contain any deprecated nodes, then this leaf SHOULD be
set to true.";
      }
      leaf obsolete-nodes-present {
        type boolean;
        default "false";
        description
          "A boolean that indicates whether or not this server implements obsole
te nodes.
           The value of this leaf SHOULD be false; and if so, the server MUST NO
```

```
T implement
           nodes within this module. If this leaf is true, then all nodes in thi
s module MUST
           be implemented as documented in the module.  Any variation of this MU
ST be listed as
           deviations.  If a module does not currently contain any obsolete node
s, then this
```

```
           leaf SHOULD be set to true.";
      }
    }
    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module/yanglib:sub
module" {
      description
        "Augmentations for the ietf-yang-library module/submodule to support sem
antic versioning.";
      leaf submodule-version {
        type string {
          pattern '[0-9]+\.[0-9]+\.[0-9]+';
        }
        description
          "The semantic version for this submodule in MAJOR.MINOR.PATCH format.
 This version
           must match the semver:module-version value in specific revision of th
e submodule
           loaded in this module-set.";
      }
    }
  }
<CODE ENDS>
```

4. Contributors

- o Anees Shaikh, Google
- o Rob Shakir, Google

5. Security Considerations

The document does not define any new protocol or data model. There are no security impacts.

6. IANA Considerations

6.1. YANG Module Registrations

The following YANG module is requested to be registred in the "IANA Module Names" registry:

The ietf-semver module:

- o Name: ietf-semver
- o XML Namespace: urn:ietf:params:xml:ns:yang:ietf-semver
- o Prefix: semver
- o Reference: [RFCXXXX]

7. References

7.1. Normative References

[I-D.verdt-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", draft-verdt-netmod-yang-versioning-reqs-00 (work in progress), July 2018.

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

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

7.2. Informative References

[I-D.clacla-netmod-model-catalog]
Clarke, J. and B. Claise, "YANG module for yangcatalog.org", draft-clacla-netmod-model-catalog-03 (work in progress), April 2018.

[I-D.claise-semver]
Claise, B., Barnes, R., and J. Clarke, "Semantic Versioning and Structure for IETF Specifications", draft-claise-semver-02 (work in progress), January 2018.

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-06 (work in progress), April 2018.

[I-D.openconfig-netmod-model-catalog]
Shaikh, A., Shakir, R., and K. D'Souza, "Catalog and registry for YANG models", draft-openconfig-netmod-model-catalog-02 (work in progress), March 2017.

[openconfigsemver]
"Semantic Versioning for Openconfig Models", <http://www.openconfig.net/docs/semver/>.

[semver] "Semantic Versioning 2.0.0", <https://www.semver.org>.

[yangcatalog]
"YANG Catalog", <https://yangcatalog.org>.

Appendix A. Appendix

A.1. Open Issues: Requirements to be Addressed

There are a few requirements of [I-D.verdt-netmod-yang-versioning-reqs] still to be addressed. These include the following:

- o A solution is required for client compatibility to address requirements 3.1 and 3.2 from [I-D.verdt-netmod-yang-versioning-reqs]. This could include adding "module sets" support to ietf-yang-library where the client can choose one set with which to use.

- o A solution for instance data to satisfy requirement 5.3 of [I-D.verdt-netmod-yang-versioning-reqs] is also required.

- o While it is believed one could work within this semver scheme to support multiple parallel trains of development within a given YANG module, some thought should be given to how this would work in support of optional requirement 4.4 of [I-D.verdt-netmod-yang-versioning-reqs].

- o While not mandatory, requirement 2.2 of [I-D.verdt-netmod-yang-versioning-reqs] looks to provide a way to determine, at the node level, whether or not changes have occurred between revisions of a given YANG module. This may require application of semver at the node level.

A.2. Open Issues

Additionally, there are a few open issues to be discussed and settled. These include the following:

- o Do we need a new version of YANG?
  While eventually this will fold into a new version, the belief is this solution can work with extensions alone with an update to the [RFC7950] text concerning module updates.

- o Should IETF/IANA officially generate derived semantic versions for their own modules? As they are the owner of the modules it should be their responsibility, but how to document it? Note that next round of funding for the yangcatalog.org could help develop the perfect derived-semantic-version toolset

- o We could consider a new naming convention for module files. Today, module files are named using a module@revision.yang notation. We could consider module%semver.yang or module#version.yang variants. Re-using the '@' for version is not ideal, so another separator character should be used. In this manner, both version and revision could be used.

Authors' Addresses

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Joe Clarke
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com

Balazs Lengyel
Ericsson
Magyar Tudosok Korutja
1117 Budapest
Hungary

Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
United States of America

Email: kd6913@att.com

NETMOD D. Ding
Internet-Draft F. Zheng
Intended status: Standards Track Huawei
Expires: April 29, 2018 October 26, 2017

YANG Data Model for ARP
draft-ding-netmod-arp-yang-model-00

Abstract

This document defines a YANG data model to describe Address Resolution Protocol (ARP) configurations. It is intended this model be used by service providers who manipulate devices from different vendors in a standard way.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a YANG [RFC6020] data model for Address Resolution Protocol [RFC826] implementation and identification of some common properties within a device containing a Network Configuration Protocol (NETCONF) server. Devices that are managed by NETCONF and perhaps other mechanisms have common properties that need to be configured and monitored in a standard way.

The data model convers configuration of system parameters of ARP, such as static ARP entries, timeout for dynamic ARP entries, interface ARP, proxy ARP, and so on. It also provides information about running state of ARP implementations.

1.1. Terminology

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].

The following terms are defined in [RFC6241] and are not redefined here:

- o client
- o configuration data
- o server

- o state data

1.2. Tree Diagrams

A simplified graphical representation of the data model is presented in Section 3.

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

- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).

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

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

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

2. Problem Statement

This document defines a YANG [RFC7950] configuration data model that may be used to configure the ARP feature running on a system. YANG models can be used with network management protocols such as NETCONF [RFC6241] to install, manipulate, and delete the configuration of network devices.

The data model makes use of the YANG "feature" construct which allows implementations to support only those ARP features that lie within their capabilities. It is intended this model be used by service providers who manipulate devices from different vendors in a standard way.

This module can be used to configure the ARP applications for discovering the link layer address associated with a given Internet layer address.

3. Design of the Data Model

This data model intends to describe the processing that a protocol finds the hardware address, also known as Media Access Control (MAC) address, of a host from its known IP address. These tasks include, but are not limited to, adding a static entry in the ARP cache, configuring ARP cache entry timeout, and clearing dynamic entries from the ARP cache.

This data model has one top level container, ARP, which consists of several second level containers. Each of these second level containers describes a particular category of ARP handling, such as defining static mapping between an IP address (32-bit address) and a Media Access Control (MAC) address (48-bit address).

```
module: ietf-arp
   +--rw arp
      +--rw arp-static-tables
      |  +--rw arp-static-table* [vrf-name ip-address]
      |     +--rw vrf-name       arp:routing-instance-ref
      |     +--rw ip-address     inet:ipv4-address-no-zone
      |     +--rw mac-address    yang:mac-address
      |     +--rw if-name?       leafref
      +--rw arp-interfaces
      |  +--rw arp-interface* [if-name]
      |     +--rw if-name                     leafref
      |     +--rw expire-time?                uint32
      |     +--rw arp-learn-disable?          boolean
      |     +--rw proxy-enable?               boolean
      |     +--rw probe-interval?             uint8
      |     +--rw probe-times?                uint8
      |     +--rw probe-unicast?              boolean
      |     +--rw arp-gratuitous?             boolean
      |     +--rw arp-gratuitous-interval?    uint32
      |     +--rw arp-gratuitous-drop?        boolean
      |     +--rw arp-if-limits
      |        +--rw arp-if-limit* [vlan-id]
      |           +--rw vlan-id            uint16
      |           +--rw limit-number       uint32
      |           +--rw threshold-value?   uint32
      +--ro arp-tables
      |  +--ro arp-table* [vrf-name ip-address]
      |     +--ro vrf-name       arp:routing-instance-ref
      |     +--ro ip-address     inet:ipv4-address-no-zone
      |     +--ro mac-address?   yang:mac-address
      |     +--ro expire-time?   uint32
      |     +--ro if-name?       leafref
      +--ro arp-statistics
         +--ro global-statistics*
         |  +--ro requests-received?     uint32
         |  +--ro replies-received?      uint32
         |  +--ro gratuitous-received?   uint32
         |  +--ro requests-sent?         uint32
         |  +--ro replies-sent?          uint32
         |  +--ro gratuitous-sent?       uint32
         |  +--ro drops-received?        uint32
```

```
            |  +--ro total-received?        uint32
            |  +--ro total-sent?            uint32
            |  +--ro arp-dynamic-count?     uint32
            |  +--ro arp-static-count?      uint32
            +--ro arp-if-statistics* [if-name]
               +--ro if-name                 leafref
               +--ro requests-received?     uint32
               +--ro replies-received?      uint32
               +--ro gratuitous-received?   uint32
               +--ro requests-sent?         uint32
               +--ro replies-sent?          uint32
               +--ro gratuitous-sent?       uint32
```

4. YANG Module

This section presents the YANG module for the ARP data model defined in this document.

```
<CODE BEGINS> file "ietf-arp@2017-10-18.yang"
module ietf-arp {
  namespace "urn:ietf:params:xml:ns:yang:ietf-arp";
  prefix arp;

 // import some basic types

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-yang-types {
    prefix yang;
  }

  import ietf-interfaces {
    prefix if;
  }

  import ietf-network-instance {
    prefix ni;
  }
  organization
    "IETF Netmod (Network Modeling) Working Group";
  contact
    "WG Web: <http://tools.ietf.org/wg/netmod/>
     WG List: <mailto: netmod@ietf.org>
```

```
     Editor: Xiaojian Ding
                   dingxiaojian1@huawei.com
     Editor: Feng Zheng
                   habby.zheng@huawei.com";
  description
    "Address Resolution Protocol (ARP) management, which includes
        static ARP configuration, dynamic ARP learning, ARP entry query,
        and packet statistics collection.";

  revision 2017-10-18 {
    description
      "Init revision";
    reference
      "RFC XXX: ARP (Address Resolution Protocol) YANG data model.";
  }

/*grouping*/

  grouping arp-prob-grouping {
    description
      "Common configuration for all ARP probe.";
    leaf probe-interval {
      type uint8 {
        range "1..5";
      }
          units "second";
      description
        "Interval for detecting dynamic ARP entries.";
    }
    leaf probe-times {
      type uint8 {
        range "0..10";
      }
      description
        "Number of aging probe attempts for a dynamic ARP entry. If
    a device does not receive an ARP reply message after the number
                of aging probe attempts reaches a specified number, the
                dynamic ARP entry is deleted.";
    }
    leaf probe-unicast {
      type boolean;
      default "false";
      description
        "Send unicast ARP aging probe messages for a dynamic ARP
                entry.";
    }
  }
```

```
  grouping arp-gratuitous-grouping {
    description
      "Configure gratuitous ARP.";
    leaf arp-gratuitous {
      type boolean;
      default "false";
      description
      "Enable or disable sending gratuitous-arp packet on
          interface.";
    }
    leaf arp-gratuitous-interval {
      type uint32 {
        range "1..86400";
      }
          units "second";
      description
        "The interval of sending gratuitous-arp packet on the
                interface.";
    }
    leaf arp-gratuitous-drop {
      type boolean;
      default "false";
      description
      "Drop the receipt of gratuitous ARP packets on the interface.";
    }
  }

  grouping arp-statistics-grouping {
    description "IP ARP statistics information";
    leaf requests-received {
      type uint32;
      description "Total ARP requests received";
    }
    leaf replies-received {
      type uint32;
      description "Total ARP replies received";
    }
    leaf gratuitous-received {
      type uint32;
      description "Total gratuitous ARP received";
    }
    leaf requests-sent {
      type uint32;
      description "Total ARP requests sent";
    }
    leaf replies-sent {
      type uint32;
      description "Total ARP replies sent";
```

```
    }
    leaf gratuitous-sent {
      type uint32;
      description "Total gratuituous ARP sent";
    }
  }

  /* Typedefs */

  typedef routing-instance-ref {
    type leafref {
      path "/ni:network-instances/ni:network-instance/ni:name";
    }
    description
      "This type is used for leafs that reference a routing instance
          configuration.";
  }

  /* Configuration data nodes */

  container arp {
    description
      "Address Resolution Protocol (ARP) management, which includes
          static ARP configuration, dynamic ARP learning, ARP entry
          query, and packet statistics collection.";

    container arp-static-tables {
      description
        "List of static ARP configurations.";
      list arp-static-table {
        key "vrf-name ip-address";
        description
          "Static ARP table. By default, the system ARP table is
                  empty, and address mappings are implemented by dynamic
                  ARP.";
        leaf vrf-name {
          type arp:routing-instance-ref;
          description
            "Name of a VPN instance. This parameter is used to
                        support the VPN feature. If this parameter is
                        set, it indicates that the ARP entry is in the
                        associated VLAN.";
        }
        leaf ip-address {
          type inet:ipv4-address-no-zone;
          description
            "IP address, in dotted decimal notation.";
        }
```

```
        leaf mac-address {
          type yang:mac-address;
          mandatory true;
          description
            "MAC address in the format of H-H-H, in which H is
                        a hexadecimal number of 1 to 4 bits. ";
        }
        leaf if-name {
          type leafref {
            path "/if:interfaces/if:interface/if:name";
          }
          description
            "Name of the ARP outbound interface.";
        }
      }
    }//End of arp-static-tables

    container arp-interfaces {
      description
        "List of ARP Interface configurations.";
      list arp-interface {
        key "if-name";
        description
          "ARP interface configuration, including the aging time,
                probe interval, number of aging probe attempts, ARP
                learning status, and ARP proxy.";
        leaf if-name {
          type leafref {
            path "/if:interfaces/if:interface/if:name";
          }
          description
            "Name of the interface that has learned dynamic ARP
                        entries.";
        }
        leaf expire-time {
          type uint32 {
            range "60..86400";
          }
                units "second";
          description
            "Aging time of a dynamic ARP entry.";
        }
        leaf arp-learn-disable {
          type boolean;
          default "false";
          description
            "Whether dynamic ARP learning is disabled. If the value
                        is True, dynamic ARP learning is disabled. If the value
```

```
                        is False, dynamic ARP learning is enabled.";
        }
        leaf proxy-enable {
          type boolean;
          default "false";
          description
            "Enable proxy ARP.";
        }
        uses arp-prob-grouping;
        uses arp-gratuitous-grouping;

        container arp-if-limits {
          description
            "Maximum number of dynamic ARP entries that an interface
                        can learn.";
          list arp-if-limit {
            key "vlan-id";
            description
              "Maximum number of dynamic ARP entries that an
                          interface can learn. If the number of ARP entries that
                          an interface can learn changes and the number of the
                          learned ARP entries exceeds the changed value, the
                          interface cannot learn additional ARP entries. The
                          system prompts you to delete the excess ARP entries.";
             leaf vlan-id {
              type uint16 {
                range "0..4094";
              }
              description
                "ID of the VLAN where ARP learning is restricted.
                                This parameter can be set only on Layer 2 interf
aces
                                and sub-interfaces. Ethernet, GE, VE, and Eth-Tr
unk
                                interfaces can be both Layer 3 and Layer 2
                                interfaces. When they work in Layer 3 mode, they
                                cannot have VLANs configured. When they work in
Layer
                                2 mode, they must have VLANs configured. Etherne
t,
                                GE, and Eth-Trunk sub-interfaces can be both com
mon
                                and QinQ sub-interfaces. ";
            }
            leaf limit-number {
              type uint32 {
                range "1..65536";
              }
              mandatory true;
              description
                "Maximum number of dynamic ARP entries that an
                                interface can learn.";
            }
```

```
            leaf threshold-value {
              type uint32 {
                range "60..100";
              }
              must "not(not(../limit-number))"{
                          description
              "Upper boundary must be higher than lower boundary.";
                          }
              description
                "Alarm-Threshold for maximum number of ARP entries
                                that an interface can learn.";
            }
          }
        }//End of arp-if-limits
      }
    }// End of arp-interfaces

    container arp-tables {
        config false;
      description
        "List of ARP entries that can be queried.";
      list arp-table {
        key "vrf-name ip-address";
        description
          "Query ARP entries, including static, dynamic, and
                  interface-based ARP entries.";
        leaf vrf-name {
          type arp:routing-instance-ref;
          description
            "Name of the VPN instance to which an ARP entry
                        belongs.";
        }
        leaf ip-address {
          type inet:ipv4-address-no-zone;
          description
            "IP address, in dotted decimal notation.";
        }
        leaf mac-address {
          type yang:mac-address;
          description
            "MAC address in the format of H-H-H, in which H is a
                        hexadecimal number of 1 to 4 bits. ";
        }
        leaf expire-time {
          type uint32 {
            range "1..1440";
          }
          description
```

```
            "Aging time of a dynamic ARP entry. ";
        }
        leaf if-name {
          type leafref {
            path "/if:interfaces/if:interface/if:name";
          }
          description
            "Type and number of the interface that has learned ARP
                        entries.";
        }
      }
    }//End of arp-tables

    container arp-statistics {
      config false;
      description
        "List of ARP packet statistics.";
      list global-statistics {
        description
          "ARP packet statistics.";
        uses arp-statistics-grouping;
        leaf drops-received {
          type uint32 {
            range "0..4294967294";
          }
          description
            "Number of ARP packets discarded.";
        }
        leaf total-received {
          type uint32 {
            range "0..4294967294";
          }
          description
            "Total number of ARP received packets.";
        }
        leaf total-sent {
          type uint32 {
            range "0..4294967294";
          }
          description
            "Total number of ARP sent packets.";
        }
        leaf arp-dynamic-count {
          type uint32 {
            range "0..4294967294";
          }
          description
            "Number of dynamic ARP count.";
```

```
        }
        leaf arp-static-count {
          type uint32 {
            range "0..4294967294";
          }
          description
            "Number of static ARP count.";
        }
      }
      list arp-if-statistics {
        key "if-name";
        description
          "ARP statistics on interfaces. ARP statistics on all
                  interfaces are displayed in sequence.";
        leaf if-name {
          type leafref {
            path "/if:interfaces/if:interface/if:name";
          }
          description
            "Name of an interface where ARP statistics to be
                        displayed reside.";
        }
        uses arp-statistics-grouping;
      }
    }// End of arp-statistics
  }
}
<CODE ENDS>
```

5. Data Model Examples

This section presents a simple but complete example of configuring static ARP entries and interfaces, based on the YANG module specified in Section 4.

5.1. Static ARP entries

Requirement:
Enable static ARP entry configuration.

```
   <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <arp xmlns="urn:ietf:params:xml:ns:yang:ietf-arp">
         <arp-static-tables>
                <vrf-name> __public__ </vrf-name>
                    <ip-address> 10.2.2.3 </ip-address>
                    <mac-address> 00e0-fc01-0000 </mac-address>
                    <if-name> GE1/0/1 </if-name>
             </arp-static-tables>
      </arp>
```

5.2. ARP interfaces

Requirement:
Enable static ARP interface configuration.

```
   <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <arp xmlns="urn:ietf:params:xml:ns:yang:ietf-arp">
         <arp-interfaces>
                <if-name> GE1/0/1 </if-name>
                <expire-time>1200</expire-time>
                    <arp-learn-disable>false</arp-learn-disable>
                <proxy-enable>false</proxy-enable>
                    <probe-interval>5</probe-interval>
                    <probe-times>3</probe-times>
                    <probe-unicast>false</probe-unicast>
                    <arp-gratuitous>false</arp-gratuitous>
                <arp-gratuitous-interval>60</arp-gratuitous-interval>
                    <arp-gratuitous-drop>false</arp-gratuitous-drop>
                    <arp-if-limits>
                       <vlan-id>3</vlan-id>
                       <limit-number>65535</limit-number>
                       <threshold-value>80</threshold-value>
                <</arp-if-limits>
          </arp-interfaces>
      </arp>
```

6. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

These are the subtrees and data nodes and their sensitivity/ vulnerability:

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.

7. Conclusions

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

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

[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>.

8.2. Informative References

[RFC0826] Plummer, D., "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November 1982, <https://www.rfc-editor.org/info/rfc826>.

[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>.

[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

Xiaojian Ding
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: dingxiaojian1@huawei.com

Feng Zheng
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: habby.zheng@huawei.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: April 6, 2018

M. Jethanandani
Cisco Systems, Inc
L. Huang
General Electric
S. Agarwal
Cisco Systems, Inc.
D. Blair
Cisco Systems, INc

October 03, 2017

Network Access Control List (ACL) YANG Data Model
draft-ietf-netmod-acl-model-14

Abstract

This document describes a data model of Access Control List (ACL) basic building blocks.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. Please note that no other RFC Editor instructions are specified anywhere else in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements

- "XXXX" --> the assigned RFC value for this draft both in this draft and in the YANG models under the revision statement.
- Revision date in model needs to get updated with the date the draft gets approved. The date also needs to get reflected on the line with <CODE BEGINS>.

Status of This Memo

Table of Contents

1. Introduction

Access Control List (ACL) is one of the basic elements used to configure device forwarding behavior. It is used in many networking technologies such as Policy Based Routing, Firewalls etc.

An ACL is an ordered set of rules that is used to filter traffic on a networking device. Each rule is represented by an Access Control Entry (ACE).

Each ACE has a group of match criteria and a group of action criteria.

The match criteria consist of a tuple of packet header match criteria and can have metadata match criteria as well.

- o Packet header matches apply to fields visible in the packet such as address or class of service or port numbers.

- o In case vendor supports it, metadata matches apply to fields associated with the packet but not in the packet header such as input interface or overall packet length

The actions specify what to do with the packet when the matching criteria is met. These actions are any operations that would apply to the packet, such as counting, policing, or simply forwarding.The list of potential actions is endless depending on the capabilities of the networked devices.

Access Control List is also widely knowns as ACL (pronounce as [ak-uh l]) or Access List. In this document, Access Control List, ACL and Access List are used interchangeably.

The matching of filters and actions in an ACE/ACL are triggered only after application/attachment of the ACL to an interface, VRF, vty/tty session, QoS policy, routing protocols amongst various other config attachment points. Once attached, it is used for filtering traffic using the match criteria in the ACE’s and taking appropriate action(s) that have been configured against that ACE. In order to apply an ACL to any attachment point, vendors would have to augment the ACL YANG model.

1.1. Definitions and Acronyms

ACE: Access Control Entry

ACL: Access Control List

DSCP: Differentiated Services Code Point

ICMP: Internet Control Message Protocol

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

MAC: Media Access Control

TCP: Transmission Control Protocol

2. Problem Statement

This document defines a YANG [RFC6020] data model for the configuration of ACLs. It is very important that model can be easily used by applications/attachments.

ACL implementations in every device may vary greatly in terms of the filter constructs and actions that they support. Therefore this draft proposes a model that can be augmented by standard extensions and vendor proprietary models.

3. Understanding ACL's Filters and Actions

Although different vendors have different ACL data models, there is a common understanding of what access control list (ACL) is. A network system usually have a list of ACLs, and each ACL contains an ordered list of rules, also known as access list entries - ACEs. Each ACE has a group of match criteria and a group of action criteria. The match criteria consist of packet header matching. It as also possible for ACE to match on metadata, if supported by the vendor. Packet header matching applies to fields visible in the packet such as address or class of service or port numbers. Metadata matching applies to fields associated with the packet, but not in the packet header such as input interface, packet length, or source or destination prefix length. The actions can be any sort of operation from logging to rate limiting or dropping to simply forwarding. Actions on the first matching ACE are applied with no processing of subsequent ACEs.

The model also includes a container to hold overall operational state for each ACL and operational state for each ACE. One ACL can be applied to multiple targets within the device, such as interfaces of a networked device, applications or features running in the device, etc. When applied to interfaces of a networked device, the ACL is applied in a direction which indicates if it should be applied to packet entering (input) or leaving the device (output). An example in the appendix shows how to express it in YANG model.

This draft tries to address the commonalities between all vendors and create a common model, which can be augmented with proprietary models. The base model is simple and with this design we hope to achieve enough flexibility for each vendor to extend the base model. The use of feature statements in the document allows vendors to advertise match rules they support.

3.1. ACL Modules

There are two YANG modules in the model. The first module, "ietf-access-control-list", defines generic ACL aspects which are common to all ACLs regardless of their type or vendor. In effect, the module can be viewed as providing a generic ACL "superclass". It imports the second module, "ietf-packet-fields". The match container in "ietf-access-control-list" uses groupings in "ietf-packet-fields". The combination of if-feature checks and must statements allow for the selection of relevant match fields that a user can define rules for.

If there is a need to define new "matches" choice, such as IPFIX [RFC5101], the container "matches" can be augmented.

For a reference to the annotations used in the diagram below, see YANG Tree Diagrams [I-D.ietf-netmod-yang-tree-diagrams].

```
module: ietf-access-control-list
    +--rw access-lists
       +--rw acl* [acl-type acl-name]
       |  +--rw acl-name    string
       |  +--rw acl-type    acl-type
       |  +--rw aces
       |     +--rw ace* [rule-name]
       |        +--rw rule-name          string
       |        +--rw matches
       |        |  +--rw l2-acl {l2-acl}?
       |        |  |  +--rw destination-mac-address?       yang:mac-ad
dress
       |        |  |  +--rw destination-mac-address-mask?   yang:mac-ad
dress
       |        |  |  +--rw source-mac-address?            yang:mac-ad
dress
       |        |  |  +--rw source-mac-address-mask?       yang:mac-ad
dress
       |        |  |  +--rw ethertype?                     eth:etherty
```

```
pe
       |         |  +--rw ipv4-acl {ipv4-acl}?
       |         |  |  +--rw dscp?                          inet:dscp
       |         |  |  +--rw ecn?                           uint8
       |         |  |  +--rw length?                        uint16
       |         |  |  +--rw ttl?                           uint8
       |         |  |  +--rw protocol?                      uint8
       |         |  |  +--rw source-port-range!
       |         |  |  |  +--rw lower-port    inet:port-number
       |         |  |  |  +--rw upper-port?   inet:port-number
       |         |  |  |  +--rw operation?    operator
       |         |  |  +--rw destination-port-range!
       |         |  |  |  +--rw lower-port    inet:port-number
       |         |  |  |  +--rw upper-port?   inet:port-number
       |         |  |  |  +--rw operations?   operator
       |         |  |  +--rw ihl?                           uint8
       |         |  |  +--rw flags?                         bits
       |         |  |  +--rw offset?                        uint16
       |         |  |  +--rw identification?                uint16
       |         |  |  +--rw destination-ipv4-network?   inet:ipv4-prefi
x
       |         |  |  +--rw source-ipv4-network?        inet:ipv4-prefi
x
       |         |  +--rw ipv6-acl {ipv6-acl}?
       |         |  |  +--rw dscp?                          inet:dscp
       |         |  |  +--rw ecn?                           uint8
       |         |  |  +--rw length?                        uint16
       |         |  |  +--rw ttl?                           uint8
       |         |  |  +--rw protocol?                      uint8
       |         |  |  +--rw source-port-range!
       |         |  |  |  +--rw lower-port    inet:port-number
       |         |  |  |  +--rw upper-port?   inet:port-number
       |         |  |  |  +--rw operation?    operator
       |         |  |  +--rw destination-port-range!
       |         |  |  |  +--rw lower-port    inet:port-number
       |         |  |  |  +--rw upper-port?   inet:port-number
       |         |  |  |  +--rw operations?   operator
       |         |  |  +--rw next-header?                   uint8
       |         |  |  +--rw destination-ipv6-network?   inet:ipv6-prefi
x
       |         |  |  +--rw source-ipv6-network?        inet:ipv6-prefi
x
       |         |  |  +--rw flow-label?                 inet:ipv6-flow-
label
       |         |  +--rw l2-l3-ipv4-acl {mixed-ipv4-acl}?
       |         |  |  +--rw destination-mac-address?        yang:mac-ad
dress
       |         |  |  +--rw destination-mac-address-mask?   yang:mac-ad
```

```
 dress
        |        |  |  +--rw source-mac-address?            yang:mac-ad
 dress
        |        |  |  +--rw source-mac-address-mask?       yang:mac-ad
 dress
        |        |  |  +--rw ethertype?                     eth:etherty
 pe
        |        |  |  +--rw dscp?                          inet:dscp
        |        |  |  +--rw ecn?                           uint8
        |        |  |  +--rw length?                        uint16
        |        |  |  +--rw ttl?                           uint8
        |        |  |  +--rw protocol?                      uint8
        |        |  |  +--rw source-port-range!
        |        |  |  |  +--rw lower-port    inet:port-number
        |        |  |  |  +--rw upper-port?   inet:port-number
        |        |  |  |  +--rw operation?    operator
        |        |  |  +--rw destination-port-range!
        |        |  |  |  +--rw lower-port    inet:port-number
        |        |  |  |  +--rw upper-port?   inet:port-number
        |        |  |  |  +--rw operations?   operator
        |        |  |  +--rw ihl?                           uint8
        |        |  |  +--rw flags?                         bits
        |        |  |  +--rw offset?                        uint16
        |        |  |  +--rw identification?                uint16
        |        |  |  +--rw destination-ipv4-network?      inet:ipv4-p
 refix
        |        |  |  +--rw source-ipv4-network?           inet:ipv4-p
 refix
        |        |  +--rw l2-l3-ipv6-acl {mixed-ipv6-acl}?
        |        |  |  +--rw destination-mac-address?       yang:mac-ad
 dress
        |        |  |  +--rw destination-mac-address-mask?  yang:mac-ad
 dress
        |        |  |  +--rw source-mac-address?            yang:mac-ad
 dress
        |        |  |  +--rw source-mac-address-mask?       yang:mac-ad
 dress
        |        |  |  +--rw ethertype?                     eth:etherty
 pe
        |        |  |  +--rw dscp?                          inet:dscp
        |        |  |  +--rw ecn?                           uint8
        |        |  |  +--rw length?                        uint16
        |        |  |  +--rw ttl?                           uint8
        |        |  |  +--rw protocol?                      uint8
        |        |  |  +--rw source-port-range!
        |        |  |  |  +--rw lower-port    inet:port-number
        |        |  |  |  +--rw upper-port?   inet:port-number
        |        |  |  |  +--rw operation?    operator
```

```
        |         |  |  +--rw destination-port-range!
        |         |  |  |  +--rw lower-port    inet:port-number
        |         |  |  |  +--rw upper-port?   inet:port-number
        |         |  |  |  +--rw operations?   operator
        |         |  |  +--rw next-header?                  uint8
        |         |  |  +--rw destination-ipv6-network?     inet:ipv6-p
 refix
        |         |  |  +--rw source-ipv6-network?          inet:ipv6-p
 refix
        |         |  |  +--rw flow-label?
        |         |  |          inet:ipv6-flow-label
        |         |  +--rw l2-l3-ipv4-ipv6-acl {l2-l3-ipv4-ipv6-acl}?
        |         |  |  +--rw destination-mac-address?      yang:mac-ad
 dress
        |         |  |  +--rw destination-mac-address-mask?   yang:mac-ad
 dress
        |         |  |  +--rw source-mac-address?           yang:mac-ad
 dress
        |         |  |  +--rw source-mac-address-mask?      yang:mac-ad
 dress
        |         |  |  +--rw ethertype?                    eth:etherty
 pe
        |         |  |  +--rw dscp?                         inet:dscp
        |         |  |  +--rw ecn?                          uint8
        |         |  |  +--rw length?                       uint16
        |         |  |  +--rw ttl?                          uint8
        |         |  |  +--rw protocol?                     uint8
        |         |  |  +--rw source-port-range!
        |         |  |  |  +--rw lower-port    inet:port-number
        |         |  |  |  +--rw upper-port?   inet:port-number
        |         |  |  |  +--rw operation?    operator
        |         |  |  +--rw destination-port-range!
        |         |  |  |  +--rw lower-port    inet:port-number
        |         |  |  |  +--rw upper-port?   inet:port-number
        |         |  |  |  +--rw operations?   operator
        |         |  |  +--rw ihl?                          uint8
        |         |  |  +--rw flags?                        bits
        |         |  |  +--rw offset?                       uint16
        |         |  |  +--rw identification?               uint16
        |         |  |  +--rw destination-ipv4-network?     inet:ipv4-p
 refix
        |         |  |  +--rw source-ipv4-network?          inet:ipv4-p
 refix
        |         |  |  +--rw next-header?                  uint8
        |         |  |  +--rw destination-ipv6-network?     inet:ipv6-p
 refix
        |         |  |  +--rw source-ipv6-network?          inet:ipv6-p
 refix
```

```
        |        |  |  +--rw flow-label?
        |        |  |          inet:ipv6-flow-label
        |        |  +--rw tcp-acl {tcp-acl}?
        |        |  |  +--rw sequence-number?          uint32
        |        |  |  +--rw acknowledgement-number?   uint32
        |        |  |  +--rw data-offset?              uint8
        |        |  |  +--rw reserved?                 uint8
        |        |  |  +--rw flags?                    bits
        |        |  |  +--rw window-size?              uint16
        |        |  |  +--rw urgent-pointer?           uint16
        |        |  |  +--rw options?                  uint32
        |        |  +--rw udp-acl {udp-acl}?
        |        |  |  +--rw length?   uint16
        |        |  +--rw icmp-acl {icmp-acl}?
        |        |  |  +--rw type?             uint8
        |        |  |  +--rw code?             uint8
        |        |  |  +--rw rest-of-header?   uint32
        |        |  +--rw any-acl! {any-acl}?
        |        |  +--rw interface?             if:interface-ref
        |        +--rw actions
        |        |       {acl-aggregate-stats or interface-acl-aggregate
 }?
        |        |  +--rw forwarding    identityref
        |        |  +--rw logging?      identityref
        |        |  +--rw icmp-off?     boolean
        |        +--ro matched-packets?   yang:counter64
        |        +--ro matched-octets?    yang:counter64
        +--rw interfaces
           +--rw interface* [interface-id]
              +--rw interface-id    if:interface-ref
              +--rw ingress
              |  +--rw acl-sets
              |     +--rw acl-set* [set-name type]
              |        +--rw set-name    -> ../../../../../../acl/acl-na
 me
              |        +--rw type        -> ../../../../../../acl/acl-ty
 pe
              |        +--rw ace* [rule-name]
              |                {interface-stats or interface-acl-aggrega
 te}?
              |           +--rw rule-name          leafref
              |           +--ro matched-packets?   yang:counter64
              |           +--ro matched-octets?    yang:counter64
              +--rw egress
                 +--rw acl-sets
                    +--rw acl-set* [set-name type]
                       +--rw set-name    -> ../../../../../../acl/acl-na
 me
```

```
                       +--rw type        -> ../../../../../../acl/acl-ty
 pe
                       +--rw ace* [rule-name]
                               {interface-stats or interface-acl-aggrega
 te}?
                          +--rw rule-name          leafref
                          +--ro matched-packets?   yang:counter64
                          +--ro matched-octets?    yang:counter64
```

4. ACL YANG Models

4.1. IETF Access Control List module

"ietf-access-control-list" is the standard top level module for access lists. The "access-lists" container stores a list of "acl". Each "acl" has information identifying the access list by a name("acl-name") and a list("access-list-entries") of rules associated with the "acl-name". Each of the entries in the list("access-list-entries"), indexed by the string "rule-name", has containers defining "matches" and "actions".

The model uses defines several ACL types in the form of identities and features. Features are used by implementors to select the ACL types the system can support. These types are implicitly inherited by the "ace", thus safeguarding against misconfiguration of "ace" types in an "acl".

The "matches" define criteria used to identify patterns in "ietf-packet-fields". The "actions" define behavior to undertake once a "match" has been identified. In addition to permit and deny for actions, a logging option allows for a match to be logged that can be used to determine which rule was matched upon. The model also defines the ability for ACL’s to be attached to a particular interface.

Statistics in the ACL can be collected for an "ace" or for an "interface". The feature statements defined for statistics can be used to determine whether statistics are being collected per "ace", per "interface" or both.

```
<CODE BEGINS> file "ietf-access-control-list@2017-10-03.yang"

module ietf-access-control-list {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-access-control-list";
  prefix acl;

  import ietf-yang-types {
```

```
    prefix yang;
  }

  import ietf-packet-fields {
    prefix packet-fields;
  }

  import ietf-interfaces {
    prefix if;
  }

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

  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
     WG List: netmod@ietf.org

     Editor: Mahesh Jethanandani
             mjethanandani@gmail.com
     Editor: Lisa Huang
             lyihuang16@gmail.com
     Editor: Sonal Agarwal
             sagarwal12@cisco.com
     Editor: Dana Blair
             dblair@cisco.com";

  description
    "This YANG module defines a component that describe the
     configuration of Access Control Lists (ACLs).

     Copyright (c) 2017 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).

     This version of this YANG module is part of RFC XXXX; see
     the RFC itself for full legal notices.";

  revision 2017-10-03 {
    description
      "Added feature and identity statements for different types
       of rule matches. Split the matching rules based on the
```

```
       feature statement and added a must statement within
       each container.";
    reference
      "RFC XXX: Network Access Control List (ACL) YANG Data Model.";
  }

  /*
   * Identities
   */

  /*
   * Forwarding actions for a packet
   */
  identity forwarding-action {
    description
      "Base identity for actions in the forwarding category";
  }

  identity accept {
    base forwarding-action;
    description
      "Accept the packet";
  }

  identity drop {
    base forwarding-action;
    description
      "Drop packet without sending any ICMP error message";
  }

  identity reject {
    base forwarding-action;
    description
      "Drop the packet and send an ICMP error message to the source";
  }

  /*
   * Logging actions for a packet
   */
  identity log-action {
    description
      "Base identity for defining the destination for logging actions";
  }

  identity log-syslog {
    base log-action;
    description
      "System log (syslog) the information for the packet";
```

```
  }

  identity log-none {
    base log-action;
    description
      "No logging for the packet";
  }

  identity acl-base {
    description
      "Base Access Control List type for all Access Control List type
       identifiers.";
  }

  identity ipv4-acl {
    base acl:acl-base;
    description
      "ACL that primarily matches on fields from the IPv4 header
       (e.g. IPv4 destination address) and layer 4 headers (e.g. TCP
       destination port).  An acl of type ipv4-acl does not contain
       matches on fields in the ethernet header or the IPv6 header.";
  }

  identity ipv6-acl {
    base acl:acl-base;
    description
      "ACL that primarily matches on fields from the IPv6 header
       (e.g. IPv6 destination address) and layer 4 headers (e.g. TCP
       destination port). An acl of type ipv6-acl does not contain
       matches on fields in the ethernet header or the IPv4 header.";
  }

  identity eth-acl {
    base acl:acl-base;
    description
      "ACL that primarily matches on fields in the ethernet header,
       like 10/100/1000baseT or WiFi Access Control List. An acl of
       type eth-acl does not contain matches on fields in the IPv4
       header, IPv6 header or layer 4 headers.";
  }

  identity mixed-l2-l3-ipv4-acl {
    base "acl:acl-base";

    description
      "ACL that contains a mix of entries that
       primarily match on fields in ethernet headers,
       entries that primarily match on IPv4 headers.
```

```
         Matching on layer 4 header fields may also exist in the
         list.";
  }

  identity mixed-l2-l3-ipv6-acl {
    base "acl:acl-base";

    description
      "ACL that contains a mix of entries that
       primarily match on fields in ethernet headers, entries
       that primarily match on fields in IPv6 headers. Matching on
       layer 4 header fields may also exist in the list.";
  }

  identity mixed-l2-l3-ipv4-ipv6-acl {
    base "acl:acl-base";

    description
      "ACL that contains a mix of entries that
       primarily match on fields in ethernet headers, entries
       that primarily match on fields in IPv4 headers, and entries
       that primarily match on fields in IPv6 headers. Matching on
       layer 4 header fields may also exist in the list.";
  }

  identity any-acl {
    base "acl:acl-base";

    description
      "ACL that can contain any pattern to match upon";
  }

  /*
   * Features
   */
  feature l2-acl {
    description
      "Layer 2 ACL supported";
  }

  feature ipv4-acl {
    description
      "Layer 3 IPv4 ACL supported";
  }

  feature ipv6-acl {
    description
      "Layer 3 IPv6 ACL supported";
```

```
  }

  feature mixed-ipv4-acl {
    description
      "Layer 2 and Layer 3 IPv4 ACL supported";
  }

  feature mixed-ipv6-acl {
    description
      "Layer 2 and Layer 3 IPv6 ACL supported";
  }

  feature l2-l3-ipv4-ipv6-acl {
    description
      "Layer 2 and any Layer 3 ACL supported.";
  }

  feature tcp-acl {
    description
      "TCP header ACL supported.";
  }

  feature udp-acl {
    description
      "UDP header ACL supported.";
  }

  feature icmp-acl {
    description
      "ICMP header ACL supported.";
  }

  feature any-acl {
    description
     "ACL for any pattern.";
  }

  /*
   * Stats Features
   */
  feature interface-stats {
    description
      "ACL counters are available and reported only per interface";
  }

  feature acl-aggregate-stats {
    description
      "ACL counters are aggregated over all interfaces, and reported
```

```
       only per ACL entry";
  }

  feature interface-acl-aggregate {
    description
      "ACL counters are reported per interface, and also aggregated
       and reported per ACL entry";
  }

  /*
   * Typedefs
   */
  typedef acl-type {
    type identityref {
      base acl-base;
    }
    description
      "This type is used to refer to an Access Control List
       (ACL) type";
  }

  typedef acl-ref {
    type leafref {
      path "/access-lists/acl/acl-name";
    }
    description
      "This type is used by data models that need to reference an
       Access Control List";
  }

  grouping interface-acl {
    description
      "Grouping for per-interface ingress ACL data";

    container acl-sets {
      description
        "Enclosing container the list of ingress ACLs on the
         interface";

      list acl-set {
        key "set-name type";
        ordered-by user;
        description
          "List of ingress ACLs on the interface";

        leaf set-name {
          type leafref {
            path "../../../../../../acl/acl-name";
```

```
          }
          description
            "Reference to the ACL set name applied on ingress";
        }

        leaf type {
          type leafref {
            path "../../../../../../acl/acl-type";
          }
          description
            "Reference to the ACL set type applied on ingress";
        }

        list ace {
          if-feature "interface-stats or interface-acl-aggregate";
          key "rule-name";
          description
            "List of access list entries(ACE)";
          leaf rule-name {
            type leafref {
              path "../../../../../../../acl/aces/ace/rule-name";
            }
            description
              "The ace rule-name";
          }
          uses acl-counters;
        }
      }
    }
  }

  grouping acl-counters {
    description
      "Common grouping for ACL counters";

    leaf matched-packets {
      type yang:counter64;
      config false;
      description
        "Count of the number of packets matching the current ACL
         entry.

         An implementation should provide this counter on a
         per-interface per-ACL-entry if possible.

         If an implementation only supports ACL counters per entry
         (i.e., not broken out per interface), then the value
         should be equal to the aggregate count across all interfaces.
```

```
         An implementation that provides counters per entry per
         interface is not required to also provide an aggregate count,
         e.g., per entry -- the user is expected to be able implement
         the required aggregation if such a count is needed.";
    }

    leaf matched-octets {
      type yang:counter64;
      config false;
      description
        "Count of the number of octets (bytes) matching the current
         ACL entry.

         An implementation should provide this counter on a
         per-interface per-ACL-entry if possible.

         If an implementation only supports ACL counters per entry
         (i.e., not broken out per interface), then the value
         should be equal to the aggregate count across all interfaces.

         An implementation that provides counters per entry per
         interface is not required to also provide an aggregate count,
         e.g., per entry -- the user is expected to be able implement
         the required aggregation if such a count is needed.";
    }
  }

  /*
   * Configuration data nodes
   */
  container access-lists {
    description
      "This is a top level container for Access Control Lists.
       It can have one or more Access Control Lists.";
    list acl {
      key "acl-type acl-name";
      description
        "An Access Control List(ACL) is an ordered list of
         Access List Entries (ACE). Each Access Control Entry has a
         list of match criteria and a list of actions.
         Since there are several kinds of Access Control Lists
         implemented with different attributes for
         different vendors, this
         model accommodates customizing Access Control Lists for
         each kind and for each vendor.";
      leaf acl-name {
        type string {
          length "1..64";
```

```
        }
        description
          "The name of access-list. A device MAY restrict the length
           and value of this name, possibly space and special
           characters are not allowed.";
      }
      leaf acl-type {
        type acl-type;
        description
          "Type of access control list. Indicates the primary intended
           type of match criteria (e.g. ethernet, IPv4, IPv6, mixed,
           etc) used in the list instance.";
      }
      container aces {
        description
          "The access-list-entries container contains
           a list of access-list-entries(ACE).";
        list ace {
          key "rule-name";
          ordered-by user;
          description
            "List of access list entries(ACE)";
          leaf rule-name {
            type string {
              length "1..64";
            }
            description
              "A unique name identifying this Access List
               Entry(ACE).";
          }

          container matches {
            description
              "The rules in this set determine what fields will be
               matched upon before any action is taken on them.
               The rules are selected based on the feature set
               defined by the server and the acl-type defined.";

            container l2-acl {
              if-feature l2-acl;
              must "derived-from(../../../../acl-type, 'acl:eth-acl')";
              uses packet-fields:acl-eth-header-fields;
              description
                "Rule set for L2 ACL.";
            }

            container ipv4-acl {
              if-feature ipv4-acl;
```

```
                must "derived-from(../../../../acl-type, " +
                     "'acl:ipv4-acl')";
              uses packet-fields:acl-ip-header-fields;
                uses packet-fields:acl-ipv4-header-fields;
              description
                "Rule set that supports IPv4 headers.";
            }

            container ipv6-acl {
              if-feature ipv6-acl;
              must "derived-from(../../../../acl-type, " +
                   "'acl:ipv6-acl')";
              uses packet-fields:acl-ip-header-fields;
              uses packet-fields:acl-ipv6-header-fields;
              description
                "Rule set that supports IPv6 headers.";
            }

            container l2-l3-ipv4-acl {
              if-feature mixed-ipv4-acl;
              must "derived-from(../../../../acl-type, " +
                   "'acl:mixed-l2-l3-ipv4-acl')";
              uses packet-fields:acl-eth-header-fields;
              uses packet-fields:acl-ip-header-fields;
              uses packet-fields:acl-ipv4-header-fields;
              description
                "Rule set that is a logical AND (&&) of l2
                 and ipv4 headers.";
            }

            container l2-l3-ipv6-acl {
              if-feature mixed-ipv6-acl;
              must "derived-from(../../../../acl-type, " +
                   "'acl:mixed-l2-l3-ipv6-acl')";
              uses packet-fields:acl-eth-header-fields;
              uses packet-fields:acl-ip-header-fields;
              uses packet-fields:acl-ipv6-header-fields;
              description
                "Rule set that is a logical AND (&&) of L2
                 && IPv6 headers.";
            }

            container l2-l3-ipv4-ipv6-acl {
              if-feature l2-l3-ipv4-ipv6-acl;
              must "derived-from(../../../../acl-type, " +
                   "'acl:mixed-l2-l3-ipv4-ipv6-acl')";
              uses packet-fields:acl-eth-header-fields;
              uses packet-fields:acl-ip-header-fields;
```

```
              uses packet-fields:acl-ipv4-header-fields;
              uses packet-fields:acl-ipv6-header-fields;
              description
                "Rule set that is a logical AND (&&) of L2
                 && IPv4 && IPv6 headers.";
            }

            container tcp-acl {
              if-feature tcp-acl;
              uses packet-fields:acl-tcp-header-fields;
              description
                "Rule set that defines TCP headers.";
            }

            container udp-acl {
              if-feature udp-acl;
              uses packet-fields:acl-udp-header-fields;
              description
                "Rule set that defines UDP headers.";
            }

            container icmp-acl {
              if-feature icmp-acl;
              uses packet-fields:acl-icmp-header-fields;
              description
                "Rule set that defines ICMP headers.";
            }

            container any-acl {
              if-feature any-acl;
              must "derived-from(../../../../acl-type, 'acl:any-acl')";
              presence "Matches any";
              description
                "Rule set that allows for a any ACL.";
            }

            leaf interface {
              type if:interface-ref;
              description
                "Interface name that is specified to
                 match upon.";
            }
          }

          container actions {
            if-feature "acl-aggregate-stats or interface-acl-aggregate";
            description
              "Definitions of action criteria for this ace entry";
```

```
                leaf forwarding {
                  type identityref {
                    base forwarding-action;
                  }
                  mandatory true;
                  description
                    "Specifies the forwarding action per ace entry";
                }

                leaf logging {
                  type identityref {
                    base log-action;
                  }
                  default log-none;
                  description
                  "Specifies the log action and destination for
                   matched packets. Default value is not to log the
                   packet.";
                }

                leaf icmp-off {
                  type boolean;
                  default "false";
                  description
                  "true indicates ICMP errors will never be generated
                   in response to an ICMP error message. false indicates
                   ICMP error will be generated.";
                }
          }
          uses acl-counters;
        }
      }
    }
    container interfaces {
      description
        "Enclosing container for the list of interfaces on which
         ACLs are set";

      list interface {
        key "interface-id";
          description
            "List of interfaces on which ACLs are set";

        leaf interface-id {
          type if:interface-ref;
          description
            "Reference to the interface id list key";
        }
```

```
        container ingress {
          uses interface-acl;
          description
            "The ACL’s applied to ingress interface";
        }
        container egress {
          uses interface-acl;
          description
            "The ACL’s applied to egress interface";
        }
      }
    }
  }
}
```

<CODE ENDS>

4.2. IETF Packet Fields module

The packet fields module defines the necessary groups for matching on fields in the packet including ethernet, ipv4, ipv6, and transport layer fields. The ’acl-type’ node determines which of these fields get included for any given ACL with the exception of TCP, UDP and ICMP header fields. Those fields can be used in conjunction with any of the above layer 2 or layer 3 fields.

Since the number of match criteria is very large, the base draft does not include these directly but references them by "uses" to keep the base module simple. In case more match conditions are needed, those can be added by augmenting choices within container "matches" in ietf-access-control-list.yang model.

<CODE BEGINS> file "ietf-packet-fields@2017-10-03.yang"

```
module ietf-packet-fields {
  namespace "urn:ietf:params:xml:ns:yang:ietf-packet-fields";
  prefix packet-fields;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-yang-types {
    prefix yang;
  }

  import ietf-ethertypes {
    prefix eth;
```

```
  }

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

  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
     WG List: netmod@ietf.org

     Editor: Mahesh Jethanandani
             mjethanandani@gmail.com
     Editor: Lisa Huang
             lyihuang16@gmail.com
     Editor: Sonal Agarwal
             agarwaso@cisco.com
     Editor: Dana Blair
             dblair@cisco.com";

  description
    "This YANG module defines groupings that are used by
    ietf-access-control-list YANG module. Their usage is not
    limited to ietf-access-control-list and can be
    used anywhere as applicable.
    Copyright (c) 2017 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).
    This version of this YANG module is part of RFC XXXX; see
    the RFC itself for full legal notices.";

  revision 2017-10-03 {
    description
      "Added header fields for TCP, UDP, and ICMP.";
    reference
      "RFC XXX: Network Access Control List (ACL) YANG Data Model.";
  }

  /*
   * Typedefs
   */
  typedef operator {
    type enumeration {
```

```
      enum lt {
        description
          "Less than.";
      }
      enum gt {
        description
          "Greater than.";
      }
      enum eq {
        description
          "Equal to.";
      }
      enum neq {
        description
          "Not equal to.";
      }
    }
    description
      "The source and destination port range definitions
       can be further qualified using an operator. An
       operator is needed only if lower-port is specified
       and upper-port is not specified. The operator
       therefore further qualifies lower-port only.";
  }

  grouping acl-transport-header-fields {
    description
      "Transport header fields";
    container source-port-range {
      presence "Enables setting source port range";
      description
        "Inclusive range representing source ports to be used.
         When only lower-port is present, it represents a single
         port and eq operator is assumed to be default.

         When both lower-port and upper-port are specified,
         it implies a range inclusive of both values.

         If no port is specified, 'any' (wildcard) is assumed.";

      leaf lower-port {
        type inet:port-number;
        mandatory true;
        description
          "Lower boundary for port.";
      }
      leaf upper-port {
        type inet:port-number;
```

```
        must ". >= ../lower-port" {
          error-message
          "The upper-port must be greater than or equal
           to lower-port";
        }
        description
          "Upper boundary for port. If it exists, the upper port
           must be greater or equal to lower-port.";
      }
      leaf operation {
        type operator;
        must "(../lower-port and not(../upper-port))" {
          error-message
            "If lower-port is specified, and an operator is also
             specified, then upper-port should not be specified.";
          description
            "If lower-port is specified, and an operator is also
             specified, then upper-port should not be specified.";
        }
        default eq;
        description
          "Operator to be applied on the lower-port.";
      }
    }

    container destination-port-range {
      presence "Enables setting destination port range";
      description
        "Inclusive range representing destination ports to be used.
         When only lower-port is present, it represents a single
         port and eq operator is assumed to be default.

         When both lower-port and upper-port are specified,
         it implies a range inclusive of both values.

         If no port is specified, ’any’ (wildcard) is assumed. ";

      leaf lower-port {
        type inet:port-number;
        mandatory true;
        description
          "Lower boundary for port.";
      }
      leaf upper-port {
        type inet:port-number;
        must ". >= ../lower-port" {
          error-message
            "The upper-port must be greater than or equal
```

```
               to lower-port";
          }
          description
            "Upper boundary for port. If existing, the upper port must
            be greater or equal to lower-port";
        }
        leaf operations {
          type operator;
          must "(../lower-port and not(../upper-port))" {
            error-message
              "If lower-port is specified, and an operator is also
               specified, then upper-port should not be specified.";
            description
              "If lower-port is specified, and an operator is also
               specified, then upper-port should not be specified.";
          }
          default eq;
          description
            "Operator to be applied on the lower-port.";
        }
      }
    }

    grouping acl-ip-header-fields {
      description
        "IP header fields common to ipv4 and ipv6";
      reference
        "RFC 791.";

      leaf dscp {
        type inet:dscp;
        description
          "Differentiated Services Code Point.";
        reference
          "RFC 2474: Definition of Differentiated services field
           (DS field) in the IPv4 and IPv6 headers.";
      }

      leaf ecn {
        type uint8 {
          range 0..3;
        }
        description
          "Explicit Congestion Notification.";
        reference
          "RFC 3168.";
      }
```

```
    leaf length {
      type uint16;
      description
        "In IPv4 header field, this field is known as the Total Length.
         Total Length is the length of the datagram, measured in octets,
         including internet header and data.

         In IPv6 header field, this field is known as the Payload
         Length, the length of the IPv6 payload, i.e. the rest of
         the packet following the IPv6 header, in octets.";
      reference
        "RFC 719, RFC 2460";
    }

    leaf ttl {
      type uint8;
      description
        "This field indicates the maximum time the datagram is allowed
         to remain in the internet system.  If this field contains the
         value zero, then the datagram must be destroyed.

         In IPv6, this field is known as the Hop Limit.";
      reference "RFC 719, RFC 2460";
    }

    leaf protocol {
      type uint8;
      description
        "Internet Protocol number.";
    }
    uses acl-transport-header-fields;
  }

  grouping acl-ipv4-header-fields {
    description
      "Fields in IPv4 header.";

    leaf ihl {
      type uint8 {
        range "5..60";
      }
      description
        "An IPv4 header field, the Internet Header Length (IHL) is
         the length of the internet header in 32 bit words, and
         thus points to the beginning of the data. Note that the
         minimum value for a correct header is 5.";
    }
```

```
    leaf flags {
      type bits {
        bit reserved {
          position 0;
          description
            "Reserved. Must be zero.";
        }
        bit fragment {
          position 1;
          description
            "Setting value to 0 indicates may fragment, while setting
             the value to 1 indicates do not fragment.";
        }
        bit more {
          position 2;
          description
            "Setting the value to 0 indicates this is the last fragment,
             and setting the value to 1 indicates more fragments are
             coming.";
        }
      }
      description
        "Bit definitions for the flags field in IPv4 header.";
    }

    leaf offset {
      type uint16 {
        range "20..65535";
      }
      description
        "The fragment offset is measured in units of 8 octets (64 bits).
         The first fragment has offset zero. The length is 13 bits";
    }

    leaf identification {
      type uint16;
      description
        "An identifying value assigned by the sender to aid in
         assembling the fragments of a datagram.";
    }

    leaf destination-ipv4-network {
      type inet:ipv4-prefix;
      description
        "Destination IPv4 address prefix.";
    }
    leaf source-ipv4-network {
      type inet:ipv4-prefix;
```

```
      description
        "Source IPv4 address prefix.";
    }
  }

  grouping acl-ipv6-header-fields {
    description
      "Fields in IPv6 header";

    leaf next-header {
      type uint8;
      description
        "Identifies the type of header immediately following the
         IPv6 header. Uses the same values as the IPv4 Protocol
         field.";
      reference
        "RFC 2460";
    }

    leaf destination-ipv6-network {
      type inet:ipv6-prefix;
      description
        "Destination IPv6 address prefix.";
    }

    leaf source-ipv6-network {
      type inet:ipv6-prefix;
      description
        "Source IPv6 address prefix.";
    }

    leaf flow-label {
      type inet:ipv6-flow-label;
      description
        "IPv6 Flow label.";
    }
    reference
      "RFC 4291: IP Version 6 Addressing Architecture
       RFC 4007: IPv6 Scoped Address Architecture
       RFC 5952: A Recommendation for IPv6 Address Text
                 Representation";
  }

  grouping acl-eth-header-fields {
    description
      "Fields in Ethernet header.";

    leaf destination-mac-address {
```

```
        type yang:mac-address;
        description
          "Destination IEEE 802 MAC address.";
      }
      leaf destination-mac-address-mask {
        type yang:mac-address;
        description
          "Destination IEEE 802 MAC address mask.";
      }
      leaf source-mac-address {
        type yang:mac-address;
        description
          "Source IEEE 802 MAC address.";
      }
      leaf source-mac-address-mask {
        type yang:mac-address;
        description
          "Source IEEE 802 MAC address mask.";
      }
      leaf ethertype {
        type eth:ethertype;
        description
          "The Ethernet Type (or Length) value represented
           in the canonical order defined by IEEE 802.
           The canonical representation uses lowercase
           characters.";
        reference
          "IEEE 802-2014 Clause 9.2";
      }
      reference
        "IEEE 802: IEEE Standard for Local and Metropolitan
         Area Networks: Overview and Architecture.";
    }

    grouping acl-tcp-header-fields {
      description
        "Collection of TCP header fields that can be used to
         setup a match filter.";

      leaf sequence-number {
        type uint32;
        description
          "Sequence number that appears in the packet.";
      }

      leaf acknowledgement-number {
        type uint32;
        description
```

```
        "The acknowledgement number that appears in the
         packet.";
    }

    leaf data-offset {
      type uint8 {
        range "5..15";
      }
      description
        "Specifies the size of the TCP header in 32-bit
         words. The minimum size header is 5 words and
         the maximum is 15 words thus giving the minimum
         size of 20 bytes and maximum of 60 bytes,
         allowing for up to 40 bytes of options in the
         header.";
    }

    leaf reserved {
      type uint8;
      description
        "Reserved for future use.";
    }

    leaf flags {
      type bits {
        bit ns {
          position 0;
          description
            "ECN-nonce concealment protection";
          reference "RFC 3540).";
        }
        bit cwr {
          position 1;
          description
            "Congestion Window Reduced (CWR) flag is set by
             the sending host to indicate that it received
             a TCP segment with the ECE flag set and had
             responded in congestion control mechanism.";
          reference "RFC 3168";
        }
        bit ece {
          position 2;
          description
            "ECN-Echo has a dual role, depending on the value
             of the SYN flag. It indicates:
             If the SYN flag is set (1), that the TCP peer is ECN
             capable. If the SYN flag is clear (0), that a packet
             with Congestion Experienced flag set (ECN=11) in IP
```

```
             header was received during normal transmission
             (added to header by RFC 3168). This serves as an
             indication of network congestion (or impending
             congestion) to the TCP sender.";
        }
        bit urg {
          position 3;
          description
            "Indicates that the Urgent pointer field is significant.";
        }
        bit ack {
          position 4;
          description
            "Indicates that the Acknowledgment field is significant.
             All packets after the initial SYN packet sent by the
             client should have this flag set.";
        }
        bit psh {
          position 5;
          description
            "Push function. Asks to push the buffered data to the
             receiving application.";
        }
        bit rst {
          position 6;
          description
            "Reset the connection.";
        }
        bit syn {
          position 7;
          description
            "Synchronize sequence numbers. Only the first packet
             sent from each end should have this flag set. Some
             other flags and fields change meaning based on this
             flag, and some are only valid for when it is set,
             and others when it is clear.";
        }
        bit fin {
          position 8;
          description
            "Last package from sender.";
        }
      }
      description
        "Also known as Control Bits. Contains 9 1-bit flags.";
    }

    leaf window-size {
```

```
      type uint16;
      description
        "The size of the receive window, which specifies
         the number of window size units (by default,
         bytes) (beyond the segment identified by the
         sequence number in the acknowledgment field)
         that the sender of this segment is currently
         willing to receive.";
    }

    leaf urgent-pointer {
      type uint16;
      description
        "This field is an offset from the sequence number
         indicating the last urgent data byte.";
    }

    leaf options {
      type uint32;
      description
        "The length of this field is determined by the
         data offset field. Options have up to three
         fields: Option-Kind (1 byte), Option-Length
         (1 byte), Option-Data (variable). The Option-Kind
         field indicates the type of option, and is the
         only field that is not optional. Depending on
         what kind of option we are dealing with,
         the next two fields may be set: the Option-Length
         field indicates the total length of the option,
         and the Option-Data field contains the value of
         the option, if applicable.";
    }
  }

  grouping acl-udp-header-fields {
    description
      "Collection of UDP header fields that can be used
       to setup a match filter.";

    leaf length {
      type uint16;
      description
        "A field that specifies the length in bytes of
         the UDP header and UDP data. The minimum
         length is 8 bytes because that is the length of
         the header. The field size sets a theoretical
         limit of 65,535 bytes (8 byte header + 65,527
         bytes of data) for a UDP datagram. However the
```

```
         actual limit for the data length, which is
         imposed by the underlying IPv4 protocol, is
         65,507 bytes (65,535 minus 8 byte UDP header
         minus 20 byte IP header).

         In IPv6 jumbograms it is possible to have
         UDP packets of size greater than 65,535 bytes.
         RFC 2675 specifies that the length field is set
         to zero if the length of the UDP header plus
         UDP data is greater than 65,535.";
    }
  }

  grouping acl-icmp-header-fields {
    description
      "Collection of ICMP header fields that can be
       used to setup a match filter.";

    leaf type {
      type uint8;
      description
        "Also known as Control messages.";
      reference "RFC 792";
    }

    leaf code {
      type uint8;
      description
        "ICMP subtype. Also known as Control messages.";
    }

    leaf rest-of-header {
      type uint32;
      description
        "Four-bytes field, contents vary based on the
         ICMP type and code.";
    }
  }
}
```

<CODE ENDS>

4.3. An ACL Example

Requirement: Deny tcp traffic from 10.10.10.1/24, destined to 11.11.11.1/24.

Here is the acl configuration xml for this Access Control List:

```
<?xml version='1.0' encoding='UTF-8'?>
  <data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <access-lists xmlns="urn:ietf:params:xml:ns:yang:
     ietf-access-control-list">
      <acl>
        <acl-name>sample-ipv4-acl</acl-name>
        <acl-type>ipv4-acl</acl-type>
        <aces>
          <ace>
            <rule-name>rule1</rule-name>
            <matches>
              <ipv4-acl>
                <protocol>tcp</protocol>
                <destination-ipv4-network>
                  11.11.11.1/24
                </destination-ipv4-network>
                <source-ipv4-network>
                  10.10.10.1/24
                </source-ipv4-network>
              </ipv4-acl>
            </matches>
            <actions>
              <packet-handling>deny</packet-handling>
            </actions>
          </ace>
        </aces>
      </acl>
    </access-lists>
  </data>
```


The acl and aces can be described in CLI as the following:

```
access-list ipv4 sample-ipv4-acl
deny tcp 10.10.10.1/24 11.11.11.1/24
```

4.4. Port Range Usage Example

When a lower-port and an upper-port are both present, it represents a range between lower-port and upper-port with both the lower-port and upper-port are included. When only a lower-port presents, it represents a single port.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>16384</lower-port>
  <upper-port>16387</upper-port>
</source-port-range>
```

This represents source ports 16384,16385, 16386, and 16387.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>16384</lower-port>
  <upper-port>65535</upper-port>
</source-port-range>
```

This represents source ports greater than/equal to 16384 and less than equal to 65535.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>21</lower-port>
</source-port-range>
```

This represents port 21.

With the following XML snippet, the configuration is specifying all ports that are not equal to 21.

```
<source-port-range>
  <lower-port>21</lower-port>
  <operations>neq</operations>
</source-port-range>
```

5. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF Access Control Model (NACM [RFC6536]) provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which 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:

/access-lists/acl/access-list-entries: This list specifies all the configured access list entries on the device. Unauthorized write access to this list can allow intruders to access and control the system. Unauthorized read access to this list can allow intruders to spoof packets with authorized addresses thereby compromising the system.

6. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-access-control-list

URI: urn:ietf:params:xml:ns:yang:ietf-packet-fields

Registrant Contact: The IESG.

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

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

name: ietf-access-control-list namespace: urn:ietf:params:xml:ns:yang:ietf-access-control-list prefix: ietf-acl reference: RFC XXXX

name: ietf-packet-fields namespace: urn:ietf:params:xml:ns:yang:ietf-packet-fields prefix: ietf-packet-fields reference: RFC XXXX

7. Acknowledgements

Alex Clemm, Andy Bierman and Lisa Huang started it by sketching out an initial IETF draft in several past IETF meetings. That draft included an ACL YANG model structure and a rich set of match filters, and acknowledged contributions by Louis Fourie, Dana Blair, Tula Kraiser, Patrick Gili, George Serpa, Martin Bjorklund, Kent Watsen, and Phil Shafer. Many people have reviewed the various earlier drafts that made the draft went into IETF charter.

Dean Bogdanovic, Kiran Agrahara Sreenivasa, Lisa Huang, and Dana Blair each evaluated the YANG model in previous drafts separately, and then worked together to created a ACL draft that was supported by different vendors. That draft removed vendor specific features, and gave examples to allow vendors to extend in their own proprietary ACL. The earlier draft was superseded with this updated draft and received more participation from many vendors.

Authors would like to thank Jason Sterne, Lada Lhotka, Juergen Schoenwalder, David Bannister, and Jeff Haas for their review of and suggestions to the draft.

8. Open Issues

- o The current model does not support the concept of "containers" used to contain multiple addresses per rule entry.

9. References

9.1. Normative References

[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>.

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

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

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

9.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-01 (work in progress), June 2017.

[RFC5101] Claise, B., Ed., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information", RFC 5101, DOI 10.17487/RFC5101, January 2008, <https://www.rfc-editor.org/info/rfc5101>.

Appendix A. Extending ACL model examples

A.1. Example of extending existing model for route filtering

With proposed modular design, it is easy to extend the model with other features. Those features can be standard features, like route filters. Route filters match on specific IP addresses or ranges of prefixes. Much like ACLs, they include some match criteria and corresponding match action(s). For that reason, it is very simple to extend existing ACL model with route filtering. The combination of a route prefix and prefix length along with the type of match determines how route filters are evaluated against incoming routes. Different vendors have different match types and in this model we are using only ones that are common across all vendors participating in this draft. As in this example, the base ACL model can be extended with company proprietary extensions, described in the next section.

```
module: example-ext-route-filter
  augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:aces/ietf-acl:ac
e/ietf-acl:matches:
    +--rw (route-prefix)?
       +--:(range)
          +--rw (ipv4-range)?
          |  +--:(v4-lower-bound)
          |  |  +--rw v4-lower-bound?   inet:ipv4-prefix
          |  +--:(v4-upper-bound)
          |     +--rw v4-upper-bound?   inet:ipv4-prefix
          +--rw (ipv6-range)?
             +--:(v6-lower-bound)
             |  +--rw v6-lower-bound?   inet:ipv6-prefix
             +--:(v6-upper-bound)
                +--rw v6-upper-bound?   inet:ipv6-prefix
```

```
  file "example-ext-route-filter@2017-10-03.yang"
  module example-ext-route-filter {
    namespace "urn:ietf:params:xml:ns:yang:example-ext-route-filter";
```

```
     prefix example-ext-route-filter;

     import ietf-inet-types {
       prefix "inet";
     }
     import ietf-access-control-list {
       prefix "ietf-acl";
     }

     organization
       "Route model group.";

     contact
       "abc@abc.com";

     description "
       This module describes route filter as a collection of
       match prefixes. When specifying a match prefix, you
       can specify an exact match with a particular route or
       a less precise match. You can configure either a
       common action that applies to the entire list or an
       action associated with each prefix.
       ";
     revision 2017-10-03 {
       description
         "Creating Route-Filter extension model based on
         ietf-access-control-list model";
       reference "Example route filter";
     }

     augment "/ietf-acl:access-lists/ietf-acl:acl/" +
             "ietf-acl:aces/ietf-acl:ace/ietf-acl:matches" {
       description "
         This module augments the matches container in the ietf-acl
         module with route filter specific actions";

       choice route-prefix{
         description "Define route filter match criteria";
         case range {
           description
             "Route falls between the lower prefix/prefix-length
              and the upperprefix/prefix-length.";
           choice ipv4-range {
             description "Defines the IPv4 prefix range";
             leaf v4-lower-bound {
               type inet:ipv4-prefix;
               description
                 "Defines the lower IPv4 prefix/prefix length";
```

```
             }
             leaf v4-upper-bound {
               type inet:ipv4-prefix;
               description
                 "Defines the upper IPv4 prefix/prefix length";
             }
           }
           choice ipv6-range {
             description "Defines the IPv6 prefix/prefix range";
             leaf v6-lower-bound {
               type inet:ipv6-prefix;
               description
                 "Defines the lower IPv6 prefix/prefix length";
             }
             leaf v6-upper-bound {
               type inet:ipv6-prefix;
               description
                 "Defines the upper IPv6 prefix/prefix length";
             }
           }
         }
       }
     }
   }
```

A.2. A company proprietary module example

Module "example-newco-acl" is an example of company proprietary model that augments "ietf-acl" module. It shows how to use 'augment' with an XPath expression to add additional match criteria, action criteria, and default actions when no ACE matches found. All these are company proprietary extensions or system feature extensions. "example-newco-acl" is just an example and it is expected from vendors to create their own proprietary models.

The following figure is the tree structure of example-newco-acl. In this example, /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace/ietf-acl:matches are augmented with two new choices, protocol-payload-choice and metadata. The protocol-payload-choice uses a grouping with an enumeration of all supported protocol values. Metadata matches apply to fields associated with the packet but not in the packet header such as overall packet length. In other example, /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace/ietf-acl:actions are augmented with new choice of actions.

```
module: example-newco-acl
  augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:aces/ietf-acl:ac
e/ietf-acl:matches:
    +--rw (protocol-payload-choice)?
    |  +--:(protocol-payload)
    |     +--rw protocol-payload* [value-keyword]
    |        +--rw value-keyword    enumeration
    +--rw (metadata)?
       +--:(packet-length)
          +--rw packet-length?      uint16
  augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:aces/ietf-acl:ac
e/ietf-acl:actions:
    +--rw (action)?
       +--:(count)
       |  +--rw count?                   string
       +--:(policer)
       |  +--rw policer?                 string
       +--:(hiearchical-policer)
          +--rw hierarchitacl-policer?   string
  augment /ietf-acl:access-lists/ietf-acl:acl:
    +--rw default-actions
       +--rw deny?   empty

  module example-newco-acl {

    yang-version 1.1;

    namespace "urn:newco:params:xml:ns:yang:example-newco-acl";

    prefix example-newco-acl;

    import ietf-access-control-list {
      prefix "ietf-acl";
    }

    organization
      "Newco model group.";

    contact
      "abc@newco.com";
    description
      "This YANG module augments IETF ACL Yang.";

    revision 2017-10-03 {
      description
        "Creating NewCo proprietary extensions to ietf-acl model";

      reference
```

```
         "RFC XXXX: Network Access Control List (ACL)
          YANG Data  Model";
     }

     augment "/ietf-acl:access-lists/ietf-acl:acl/" +
             "ietf-acl:aces/ietf-acl:ace/" +
             "ietf-acl:matches" {
       description "Newco proprietary simple filter matches";
       choice protocol-payload-choice {
         description "Newo proprietary payload match condition";
         list protocol-payload {
           key value-keyword;
           ordered-by user;
           description "Match protocol payload";
           uses match-simple-payload-protocol-value;
         }
       }

       choice metadata {
         description "Newco proprietary interface match condition";
         leaf packet-length {
           type uint16;
           description "Match on packet length";
         }
       }
     }

     augment "/ietf-acl:access-lists/ietf-acl:acl/" +
             "ietf-acl:aces/ietf-acl:ace/" +
             "ietf-acl:actions" {
       description "Newco proprietary simple filter actions";
       choice action {
         description "";
         case count {
           description "Count the packet in the named counter";
           leaf count {
             type string;
             description "";
           }
         }
         case policer {
           description "Name of policer to use to rate-limit traffic";
           leaf policer {
             type string;
             description "";
           }
         }
         case hiearchical-policer {
```

```
           description "Name of hierarchical policer to use to
                        rate-limit traffic";
           leaf hierarchitacl-policer {
             type string;
             description "";
           }
         }
       }
     }

     augment "/ietf-acl:access-lists/ietf-acl:acl" {
       description "Newco proprietary default action";
       container default-actions {
         description
           "Actions that occur if no access-list entry is matched.";
         leaf deny {
           type empty;
           description "";
         }
       }
     }

     grouping match-simple-payload-protocol-value {
       description "Newco proprietary payload";
       leaf value-keyword {
         type enumeration {
           enum icmp {
             description "Internet Control Message Protocol";
           }
           enum icmp6 {
             description "Internet Control Message Protocol Version 6";
           }
           enum range {
             description "Range of values";
           }
         }
         description "(null)";
       }
     }
   }
```

Draft authors expect that different vendors will provide their own yang models as in the example above, which is the augmentation of the base model

A.3. Linux nftables

As Linux platform is becoming more popular as networking platform, the Linux data model is changing. Previously ACLs in Linux were highly protocol specific and different utilities were used (iptables, ip6tables, arptables, ebtables), so each one had separate data model. Recently, this has changed and a single utility, nftables, has been developed. With a single application, it has a single data model for filewall filters and it follows very similarly to the ietf-access-control list module proposed in this draft. The nftables support input and output ACEs and each ACE can be defined with match and action.

The example in Section 4.3 can be configured using nftable tool as below.

```
nft add table ip filter
nft add chain filter input
nft add rule ip filter input ip protocol tcp ip saddr \
    10.10.10.1/24 drop
```

The configuration entries added in nftable would be.

```
table ip filter {
  chain input {
    ip protocol tcp ip saddr 10.10.10.1/24 drop
  }
}
```

We can see that there are many similarities between Linux nftables and IETF ACL YANG data models and its extension models. It should be fairly easy to do translation between ACL YANG model described in this draft and Linux nftables.

A.4. Ethertypes

The ACL module is dependent on the definition of ethertypes. IEEE owns the allocation of those ethertypes. This model is being included here to enable definition of those types till such time that IEEE takes up the task of publication of the model that defines those ethertypes. At that time, this model can be deprecated.

<CODE BEGINS> file "ietf-ethertypes@2017-10-03.yang"

```
module ietf-ethertypes {
  namespace "urn:ietf:params:xml:ns:yang:ietf-ethertypes";
  prefix ie;
```

```
  organization
    "IETF NETMOD (NETCONF Data Modeling Language)";

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

     Editor:   Mahesh Jethanandani
               <mjethanandani@gmail.com>";

  description
    "This module contains the common definitions for the
     Ethertype used by different modules. It is a
     placeholder module, till such time that IEEE
     starts a project to define these Ethertypes
     and publishes a standard.

     At that time this module can be deprecated.";

  revision 2017-10-03 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: IETF Ethertype YANG Data Module.";
  }

  typedef ethertype {
    type union {
      type uint16;
      type enumeration {
        enum ipv4 {
          value 2048;
          description
            "Internet Protocol version 4 (IPv4) with a
             hex value of 0x0800.";
          reference
            "RFC 791, Internet Protocol.";
        }
        enum arp {
          value 2054;
          description
            "Address Resolution Protocol (ARP) with a
             hex value of 0x0806.";
          reference
            "RFC 826 An Ethernet Address Resolution Protocol.";
        }
        enum wlan {
          value 2114;
```

```
            description
              "Wake-on-LAN. Hex value of 0x0842.";
          }
          enum trill {
            value 8947;
            description
              "Transparent Interconnection of Lots of Links.
               Hex value of 0x22F3.";
            reference
              "RFC 6325 Routing Bridges (RBridges): Base Protocol
               Specification.";
          }
          enum srp {
            value 8938;
            description
              "Stream Reservation Protocol. Hex value of
               0x22EA.";
            reference
              "IEEE 801.1Q-2011.";
          }
          enum decnet {
            value 24579;
            description
              "DECnet Phase IV. Hex value of 0x6003.";
          }
          enum rarp {
            value 32821;
            description
              "Reverse Address Resolution Protocol.
               Hex value 0x8035.";
            reference
                  "RFC 903. A Reverse Address Resolution Protocol.";
          }
          enum appletalk {
            value 32923;
            description
              "Appletalk (Ethertalk). Hex value 0x809B.";
          }
          enum aarp {
            value 33011;
            description
              "Appletalk Address Resolution Protocol. Hex value
               of 0x80F3.";
          }
          enum vlan {
            value 33024;
            description
              "VLAN-tagged frame (802.1Q) and Shortest Path
```

```
               Bridging IEEE 802.1aq with NNI compatibility.
               Hex value 0x8100.";
          reference
                "802.1Q.";

        }
        enum ipx {
          value 33079;
          description
            "Internetwork Packet Exchange (IPX). Hex value
             of 0x8137.";
        }
        enum qnx {
          value 33284;
          description
            "QNX Qnet. Hex value of 0x8204.";
        }
        enum ipv6 {
          value 34525;
          description
            "Internet Protocol Version 6 (IPv6). Hex value
             of 0x86DD.";
          reference
            "RFC 8200, 8201.";
        }
        enum efc {
          value 34824;
          description
            "Ethernet flow control using pause frames.
             Hex value of 0x8808";
          reference
            "IEEE Std. 802.1Qbb.";
        }
        enum esp {
          value 34825;
          description
            "Ethernet Slow Protocol. Hex value of 0x8809.";
          reference
            "IEEE Std. 802.3-2015";
        }
        enum cobranet {
          value 34841;
          description
            "CobraNet. Hex value of 0x";
        }
        enum mpls-unicast {
          value 34887;
          description
```

```
              "MultiProtocol Label Switch (MPLS) unicast traffic.
               Hex value of 0x8847.";
            reference
              "RFC 3031.";
          }
          enum mpls-multicast {
            value 34888;
            description
              "MultiProtocol Label Switch (MPLS) multicast traffic.
               Hex value of 0x8848.";
            reference
              "RFC 3031.";
          }
          enum pppoe-discovery {
            value 34915;
            description
              "Point-to-Point Protocol over Ethernet. Used during
               the discovery process. Hex value of 0x8863.";
            reference
              "RFC 2516.";
          }
          enum pppoe-session {
            value 34916;
            description
              "Point-to-Point Protocol over Ethernet. Used during
               session stage. Hex value of 0x8864.";
            reference
              "RFC 2516.";
          }
          enum intel-ans {
            value 34925;
            description
              "Intel Advanced Networking Services. Hex value of
               0x886D.";
          }
          enum jumbo-frames {
            value 34928;
            description
              "Jumbo frames or Ethernet frames with more than
               1500 bytes of payload, upto 9000 bytes.";
          }
          enum homeplug {
            value 34939;
            description
              "Family name for the various power line
               communications. Hex value of 0x887B.";
          }
          enum eap {
```

```
             value 34958;
             description
               "Ethernet Access Protocol (EAP) over LAN. Hex value
                of 0x888E.";
             reference
               "IEEE 802.1X";
           }
           enum profinet {
             value 34962;
             description
               "PROcess FIeld Net (PROFINET). Hex value of 0x8892.";
           }
           enum hyperscsi {
             value 34970;
             description
               "SCSI over Ethernet. Hex value of 0x889A";
           }
           enum aoe {
             value 34978;
             description
               "Advanced Technology Advancement (ATA) over Ethernet.
                Hex value of 0x88A2.";
           }
           enum ethercat {
             value 34980;
             description
               "Ethernet for Control Automation Technology (EtherCAT).
                Hex value of 0x88A4.";
           }
           enum provider-bridging {
             value 34984;
             description
               "Provider Bridging (802.1ad) and Shortest Path Bridging
                (801.1aq). Hex value of 0x88A8.";
             reference
               "IEEE 802.1ad, IEEE 802.1aq).";
           }
           enum ethernet-powerlink {
             value 34987;
             description
               "Ethernet Powerlink. Hex value of 0x88AB.";
           }
           enum goose {
             value 35000;
             description
               "Generic Object Oriented Substation Event (GOOSE).
                Hex value of 0x88B8.";
             reference
```

```
            "IEC/ISO 8802-2 and 8802-3.";
        }
        enum gse {
          value 35001;
          description
            "Generic Substation Events. Hex value of 88B9.";
          reference
            "IEC 61850.";
        }
        enum sv {
          value 35002;
          description
            "Sampled Value Transmission. Hex value of 0x88BA.";
          reference
            "IEC 61850.";
        }
        enum lldp {
          value 35020;
          description
            "Link Layer Discovery Protocol (LLDP). Hex value of
             0x88CC.";
          reference
            "IEEE 802.1AB.";
        }
        enum sercos {
          value 35021;
          description
            "Sercos Interface. Hex value of 0x88CD.";
        }
        enum wsmp {
          value 35036;
          description
            "WAVE Short Message Protocl (WSMP). Hex value of
             0x88DC.";
        }
        enum homeplug-av-mme {
          value 35041;
          description
            "HomePlug AV MME. Hex value of 88E1.";
        }
        enum mrp {
          value 35043;
          description
            "Media Redundancy Protocol (MRP). Hex value of
             0x88E3.";
          reference
            "IEC62439-2.";
        }
```

```
           enum macsec {
             value 35045;
             description
               "MAC Security. Hex value of 0x88E5.";
             reference
               "IEEE 802.1AE.";
           }
           enum pbb {
             value 35047;
             description
               "Provider Backbone Bridges (PBB). Hex value of
                0x88E7.";
             reference
               "IEEE 802.1ah.";
           }
           enum cfm {
             value 35074;
             description
               "Connectivity Fault Management (CFM). Hex value of
                0x8902.";
             reference
               "IEEE 802.1ag.";
           }
           enum fcoe {
             value 35078;
             description
               "Fiber Channel over Ethernet (FCoE). Hex value of
                0x8906.";
             reference
               "T11 FC-BB-5.";
           }
           enum fcoe-ip {
             value 35092;
             description
               "FCoE Initialization Protocol. Hex value of 0x8914.";
           }
           enum roce {
             value 35093;
             description
               "RDMA over Converged Ethernet (RoCE). Hex value of
                0x8915.";
           }
           enum tte {
             value 35101;
             description
               "TTEthernet Protocol Control Frame (TTE). Hex value
                of 0x891D.";
             reference
```

```
              "SAE AS6802.";
          }
          enum hsr {
            value 35119;
            description
              "High-availability Seamless Redundancy (HSR). Hex
               value of 0x892F.";
            reference
              "IEC 62439-3:2016.";
          }
          enum ctp {
            value 36864;
            description
              "Ethernet Configuration Test Protocol (CTP). Hex
               value of 0x9000.";
          }
          enum vlan-double-tagged {
            value 37120;
            description
              "VLAN-tagged frame with double tagging. Hex value
               of 0x9100.";
          }
        }
      }
      description
        "The uint16 type placeholder type is defined to enable
         users to manage their own ethertypes not
         covered by the module. Otherwise the module contains
         enum definitions for the more commonly used ethertypes.";
    }
  }

  <CODE ENDS>
```

Authors' Addresses

Mahesh Jethanandani
Cisco Systems, Inc

Email: mjethanandani@gmail.com

Lisa Huang
General Electric

Email: lyihuang16@gmail.com

Sonal Agarwal
Cisco Systems, Inc.

Email: agarwaso@cisco.com

Dana Blair
Cisco Systems, INc

Email: dblair@cisco.com

NETMOD WG M. Jethanandani
Internet-Draft VMware
Intended status: Standards Track S. Agarwal
Expires: May 10, 2019 Cisco Systems, Inc.
L. Huang

D. Blair
November 6, 2018

Network Access Control List (ACL) YANG Data Model
draft-ietf-netmod-acl-model-21

Abstract

This document defines a data model for Access Control List (ACL). An ACL is a user-ordered set of rules, used to configure the forwarding behavior in device. Each rule is used to find a match on a packet, and define actions that will be performed on the packet.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Access Control List (ACL) is one of the basic elements used to configure device forwarding behavior. It is used in many networking technologies such as Policy Based Routing (PBR), firewalls etc.

An ACL is an user-ordered set of rules, that is used to filter traffic on a networking device. Each rule is represented by an Access Control Entry (ACE).

Each ACE has a group of match criteria and a group of actions.

The match criteria allow for definition of packet headers and metadata, the contents of which must match the definitions.

- o Packet header matches apply to fields visible in the packet such as address or Class of Service (CoS) or port numbers.

- o In case a vendor supports it, metadata matches apply to fields associated with the packet but not in the packet header such as input interface or length of the packet as received over the wire.

The actions specify what to do with the packet when the matching criteria are met. These actions are any operations that would apply to the packet, such as counting, policing, or simply forwarding. The list of potential actions is unbounded depending on the capabilities of the networking devices.

Access Control List is also widely knowns as ACL (pronounce as [ak-uh l]) or Access List. In this document, Access Control List, ACL and Access List are used interchangeably.

The matching of filters and actions in an ACE/ACL are triggered only after the application/attachment of the ACL to an interface, VRF, vty/tty session, QoS policy, or routing protocols, amongst various other configuration attachment points. Once attached, it is used for filtering traffic using the match criteria in the ACEs and taking appropriate action(s) that have been configured against that ACE. In order to apply an ACL to any attachment point other than an interface, vendors would have to augment the ACL YANG model.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. Please note that no other RFC Editor instructions are specified anywhere else in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements

- o "XXXX" --> the assigned RFC value for this draft both in this draft and in the YANG models under the revision statement.

- o Revision date in model, in the format 2018-11-06 needs to get updated with the date the draft gets approved. The date also needs to get reflected on the line with <CODE BEGINS>.

1.1. Definitions and Acronyms

ACE: Access Control Entry

ACL: Access Control List

CoS: Class of Service

DSCP: Differentiated Services Code Point

ICMP: Internet Control Message Protocol

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

MAC: Media Access Control

PBR: Policy Based Routing

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

1.2. Terminology

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.

1.3. Tree Diagram

For a reference to the annotations used in tree diagrams included in this draft, please see YANG Tree Diagrams [RFC8340].

2. Problem Statement

This document defines a YANG 1.1 [RFC7950] data model for the configuration of ACLs. The model defines matching rules for commonly used protocols such as, Ethernet, IPv4, IPv6, TCP, UDP and ICMP. If more protocols need to be supported in the future, this base model can be augmented. An example of such an augmentation can be seen in the Appendix.

ACL implementations in every device may vary greatly in terms of the filter constructs and actions that they support. Therefore, this draft proposes a model that can be augmented by standard extensions and vendor proprietary models.

3. Understanding ACL's Filters and Actions

Although different vendors have different ACL data models, there is a common understanding of what Access Control List (ACL) is. A network system usually has a list of ACLs, and each ACL contains an ordered list of rules, also known as Access Control Entries (ACE). Each ACE has a group of match criteria and a group of actions. The match criteria allow for definition of contents of the packet headers or metadata, if supported by the vendor. Packet header matching applies to fields visible in the packet such as address or CoS or port numbers. Metadata matching applies to fields associated with the packet, but not in the packet header, such as input interface, packet length, or source or destination prefix length. The actions can be any sort of operation from logging to rate limiting or dropping to simply forwarding. Actions on the first matching ACE are applied with no processing of subsequent ACEs.

The model also includes a container to hold overall operational state for each ACL and operational state for each ACE. One ACL can be applied to multiple targets within the device, such as interface of a networking device, applications or features running in the device, etc. When applied to interfaces of a networked device, distinct ACLs are defined for the ingress (input) or egress (output) interface.

This draft tries to address the commonalities between all vendors and create a common model, which can be augmented with proprietary models. The base model is simple in design, and we hope to achieve enough flexibility for each vendor to extend the base model.

The use of feature statements in the model allows vendors to advertise match rules they are capable and willing to support. There are two sets of feature statements a device needs to advertise. The first set of feature statements specify the capability of the device. These include features such as "Device can support matching on Ethernet headers" or "Device can support matching on IPv4 headers". The second set of feature statements specify the combinations of headers the device is willing to support. These include features such as "Plain IPv6 ACL supported" or "Ethernet, IPv4 and IPv6 ACL combinations supported".

3.1. ACL Modules

There are two YANG modules in the model. The first module, "ietf-access-control-list", defines generic ACL aspects which are common to all ACLs regardless of their type or vendor. In effect, the module can be viewed as providing a generic ACL "superclass". It imports the second module, "ietf-packet-fields". The match container in "ietf-access-control-list" uses groupings in "ietf-packet-fields" to specify match fields such as port numbers or protocol. The combination of 'if-feature' checks and 'must' statements allow for the selection of relevant match fields that a user can define rules for.

If there is a need to define a new "matches" choice, such as IPFIX [RFC7011], the container "matches" can be augmented.

```
module: ietf-access-control-list
  +--rw acls
     +--rw acl* [name]
     |  +--rw name    string
     |  +--rw type?   acl-type
     |  +--rw aces
     |     +--rw ace* [name]
     |        +--rw name          string
     |        +--rw matches
     |        |  +--rw (l2)?
     |        |  |  +--:(eth)
     |        |  |     +--rw eth {match-on-eth}?
     |        |  |        +--rw destination-mac-address?
     |        |  |        |       yang:mac-address
     |        |  |        +--rw destination-mac-address-mask?
     |        |  |        |       yang:mac-address
     |        |  |        +--rw source-mac-address?
     |        |  |        |       yang:mac-address
     |        |  |        +--rw source-mac-address-mask?
     |        |  |        |       yang:mac-address
     |        |  |        +--rw ethertype?
     |        |  |                eth:ethertype
     |        |  +--rw (l3)?
     |        |  |  +--:(ipv4)
     |        |  |  |  +--rw ipv4 {match-on-ipv4}?
     |        |  |  |     +--rw dscp?
     |        |  |  |     |       inet:dscp
     |        |  |  |     +--rw ecn?
     |        |  |  |     |       uint8
     |        |  |  |     +--rw length?
     |        |  |  |     |       uint16
     |        |  |  |     +--rw ttl?
```

```
     |        |  |  |     |       uint8
     |        |  |  |     +--rw protocol?
     |        |  |  |     |       uint8
     |        |  |  |     +--rw ihl?
     |        |  |  |     |       uint8
     |        |  |  |     +--rw flags?
     |        |  |  |     |       bits
     |        |  |  |     +--rw offset?
     |        |  |  |     |       uint16
     |        |  |  |     +--rw identification?
     |        |  |  |     |       uint16
     |        |  |  |     +--rw (destination-network)?
     |        |  |  |     |  +--:(destination-ipv4-network)
     |        |  |  |     |     +--rw destination-ipv4-network?
     |        |  |  |     |             inet:ipv4-prefix
     |        |  |  |     +--rw (source-network)?
     |        |  |  |        +--:(source-ipv4-network)
     |        |  |  |           +--rw source-ipv4-network?
     |        |  |  |                   inet:ipv4-prefix
     |        |  |  +--:(ipv6)
     |        |  |     +--rw ipv6 {match-on-ipv6}?
     |        |  |        +--rw dscp?
     |        |  |        |       inet:dscp
     |        |  |        +--rw ecn?
     |        |  |        |       uint8
     |        |  |        +--rw length?
     |        |  |        |       uint16
     |        |  |        +--rw ttl?
     |        |  |        |       uint8
     |        |  |        +--rw protocol?
     |        |  |        |       uint8
     |        |  |        +--rw (destination-network)?
     |        |  |        |  +--:(destination-ipv6-network)
     |        |  |        |     +--rw destination-ipv6-network?
     |        |  |        |             inet:ipv6-prefix
     |        |  |        +--rw (source-network)?
     |        |  |        |  +--:(source-ipv6-network)
     |        |  |        |     +--rw source-ipv6-network?
     |        |  |        |             inet:ipv6-prefix
     |        |  |        +--rw flow-label?
     |        |  |                inet:ipv6-flow-label
     |        |  +--rw (l4)?
     |        |  |  +--:(tcp)
     |        |  |  |  +--rw tcp {match-on-tcp}?
     |        |  |  |     +--rw sequence-number?          uint32
     |        |  |  |     +--rw acknowledgement-number?   uint32
     |        |  |  |     +--rw data-offset?              uint8
     |        |  |  |     +--rw reserved?                 uint8
```

```
        |           |   |   |      +--rw flags?                  bits
        |           |   |   |      +--rw window-size?            uint16
        |           |   |   |      +--rw urgent-pointer?         uint16
        |           |   |   |      +--rw options?                binary
        |           |   |   |      +--rw source-port
        |           |   |   |      |  +--rw (source-port)?
        |           |   |   |      |     +--:(range-or-operator)
        |           |   |   |      |        +--rw (port-range-or-operator)?
        |           |   |   |      |           +--:(range)
        |           |   |   |      |           |  +--rw lower-port
        |           |   |   |      |           |  |       inet:port-number
        |           |   |   |      |           |  +--rw upper-port
        |           |   |   |      |           |          inet:port-number
        |           |   |   |      |           +--:(operator)
        |           |   |   |      |              +--rw operator?     operator
        |           |   |   |      |              +--rw port
        |           |   |   |      |                      inet:port-number
        |           |   |   |      +--rw destination-port
        |           |   |   |         +--rw (destination-port)?
        |           |   |   |            +--:(range-or-operator)
        |           |   |   |               +--rw (port-range-or-operator)?
        |           |   |   |                  +--:(range)
        |           |   |   |                  |  +--rw lower-port
        |           |   |   |                  |  |       inet:port-number
        |           |   |   |                  |  +--rw upper-port
        |           |   |   |                  |          inet:port-number
        |           |   |   |                  +--:(operator)
        |           |   |   |                     +--rw operator?     operator
        |           |   |   |                     +--rw port
        |           |   |   |                             inet:port-number
        |           |   |   +--:(udp)
        |           |   |   |  +--rw udp {match-on-udp}?
        |           |   |   |     +--rw length?            uint16
        |           |   |   |     +--rw source-port
        |           |   |   |     |  +--rw (source-port)?
        |           |   |   |     |     +--:(range-or-operator)
        |           |   |   |     |        +--rw (port-range-or-operator)?
        |           |   |   |     |           +--:(range)
        |           |   |   |     |           |  +--rw lower-port
        |           |   |   |     |           |  |       inet:port-number
        |           |   |   |     |           |  +--rw upper-port
        |           |   |   |     |           |          inet:port-number
        |           |   |   |     |           +--:(operator)
        |           |   |   |     |              +--rw operator?     operator
        |           |   |   |     |              +--rw port
        |           |   |   |     |                      inet:port-number
        |           |   |   |     +--rw destination-port
        |           |   |   |        +--rw (destination-port)?
```

```
        |        |  |  |           +--:(range-or-operator)
        |        |  |  |              +--rw (port-range-or-operator)?
        |        |  |  |                 +--:(range)
        |        |  |  |                 |  +--rw lower-port
        |        |  |  |                 |  |       inet:port-number
        |        |  |  |                 |  +--rw upper-port
        |        |  |  |                 |          inet:port-number
        |        |  |  |                 +--:(operator)
        |        |  |  |                    +--rw operator?     operator
        |        |  |  |                    +--rw port
        |        |  |  |                            inet:port-number
        |        |  |  +--:(icmp)
        |        |  |     +--rw icmp {match-on-icmp}?
        |        |  |        +--rw type?             uint8
        |        |  |        +--rw code?             uint8
        |        |  |        +--rw rest-of-header?   binary
        |        |  +--rw egress-interface?    if:interface-ref
        |        |  +--rw ingress-interface?   if:interface-ref
        |        +--rw actions
        |        |  +--rw forwarding    identityref
        |        |  +--rw logging?      identityref
        |        +--ro statistics {acl-aggregate-stats}?
        |           +--ro matched-packets?   yang:counter64
        |           +--ro matched-octets?    yang:counter64
        +--rw attachment-points
           +--rw interface* [interface-id] {interface-attachment}?
              +--rw interface-id    if:interface-ref
              +--rw ingress
              |  +--rw acl-sets
              |     +--rw acl-set* [name]
              |        +--rw name              -> /acls/acl/name
              |        +--ro ace-statistics* [name] {interface-stats}?
              |           +--ro name
              |           |       -> /acls/acl/aces/ace/name
              |           +--ro matched-packets?   yang:counter64
              |           +--ro matched-octets?    yang:counter64
              +--rw egress
                 +--rw acl-sets
                    +--rw acl-set* [name]
                       +--rw name              -> /acls/acl/name
                       +--ro ace-statistics* [name] {interface-stats}?
                          +--ro name
                          |       -> /acls/acl/aces/ace/name
                          +--ro matched-packets?   yang:counter64
                          +--ro matched-octets?    yang:counter64
```

4. ACL YANG Models

4.1. IETF Access Control List module

"ietf-access-control-list" module defines the "acls" container that has a list of "acl". Each "acl" has information identifying the access list by a name ("name") and a list ("aces") of rules associated with the "name". Each of the entries in the list ("aces"), indexed by the string "name", has containers defining "matches" and "actions".

The model defines several ACL types and actions in the form of identities and features. Features are used by implementors to select the ACL types the system can support and identities are used to validate the types that have been selected. These types are implicitly inherited by the "ace", thus safeguarding against misconfiguration of "ace" types in an "acl".

The "matches" define criteria used to identify patterns in "ietf-packet-fields". The choice statements within the match container allow for selection of one header within each of "l2", "l3", or "l4" headers. The "actions" define behavior to undertake once a "match" has been identified. In addition to permit and deny for actions, a logging option allows for a match to be logged that can later be used to determine which rule was matched upon. The model also defines the ability for ACLs to be attached to a particular interface.

Statistics in the ACL can be collected for an "ace" or for an "interface". The feature statements defined for statistics can be used to determine whether statistics are being collected per "ace", or per "interface".

This module imports definitions from Common YANG Data Types [RFC6991], and A YANG Data Model for Interface Management [RFC8343].

```
<CODE BEGINS> file "ietf-access-control-list@2018-11-06.yang"

module ietf-access-control-list {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-access-control-list";
  prefix acl;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991 - Common YANG Data Types.";
  }
```

```
   import ietf-packet-fields {
     prefix pf;
     reference
     "RFC XXXX - Network ACL YANG Model.";
   }

   import ietf-interfaces {
     prefix if;
     reference
       "RFC 8343 - A YANG Data Model for Interface Management.";
   }

   organization
     "IETF NETMOD (Network Modeling Language)
      Working Group";

   contact
     "WG Web: http://tools.ietf.org/wg/netmod/
      WG List: netmod@ietf.org

      Editor: Mahesh Jethanandani
              mjethanandani@gmail.com
      Editor: Lisa Huang
              lyihuang16@gmail.com
      Editor: Sonal Agarwal
              sagarwal12@gmail.com
      Editor: Dana Blair
              dblair@cisco.com";

   description
     "This YANG module defines a component that describe the
      configuration and monitoring of Access Control Lists (ACLs).

      Copyright (c) 2018 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).

      This version of this YANG module is part of RFC XXXX; see
      the RFC itself for full legal notices.";

   revision 2018-11-06 {
     description
       "Initial version.";
```

```
     reference
       "RFC XXX: Network Access Control List (ACL) YANG Data Model.";
   }

   /*
    * Identities
    */

   /*
    * Forwarding actions for a packet
    */
   identity forwarding-action {
     description
       "Base identity for actions in the forwarding category";
   }

   identity accept {
     base forwarding-action;
     description
       "Accept the packet";
   }

   identity drop {
     base forwarding-action;
     description
       "Drop packet without sending any ICMP error message";
   }

   identity reject {
     base forwarding-action;
     description
       "Drop the packet and send an ICMP error message to the source";
   }

   /*
    * Logging actions for a packet
    */
   identity log-action {
     description
       "Base identity for defining the destination for logging actions";
   }

   identity log-syslog {
     base log-action;
     description
       "System log (syslog) the information for the packet";
   }
```

```
   identity log-none {
     base log-action;
     description
       "No logging for the packet";
   }

   /*
    * ACL type identities
    */
   identity acl-base {
     description
       "Base Access Control List type for all Access Control List type
        identifiers.";
   }

   identity ipv4-acl-type {
     base acl:acl-base;
     if-feature "ipv4";
     description
       "An ACL that matches on fields from the IPv4 header
        (e.g. IPv4 destination address) and layer 4 headers (e.g. TCP
        destination port).  An acl of type ipv4 does not contain
        matches on fields in the ethernet header or the IPv6 header.";
   }

   identity ipv6-acl-type {
     base acl:acl-base;
     if-feature "ipv6";
     description
       "An ACL that matches on fields from the IPv6 header
        (e.g. IPv6 destination address) and layer 4 headers (e.g. TCP
        destination port). An acl of type ipv6 does not contain
        matches on fields in the ethernet header or the IPv4 header.";
   }

   identity eth-acl-type {
     base acl:acl-base;
     if-feature "eth";
     description
       "An ACL that matches on fields in the ethernet header,
        like 10/100/1000baseT or WiFi Access Control List. An acl of
        type ethernet does not contain matches on fields in the IPv4
        header, IPv6 header or layer 4 headers.";
   }

   identity mixed-eth-ipv4-acl-type {
     base "acl:eth-acl-type";
     base "acl:ipv4-acl-type";
```

```
     if-feature "mixed-eth-ipv4";
     description
       "An ACL that contains a mix of entries that
        match on fields in ethernet headers,
        entries that match on IPv4 headers.
        Matching on layer 4 header fields may also exist in the
        list.";
   }

   identity mixed-eth-ipv6-acl-type {
     base "acl:eth-acl-type";
     base "acl:ipv6-acl-type";
     if-feature "mixed-eth-ipv6";
     description
       "ACL that contains a mix of entries that
        match on fields in ethernet headers, entries
        that match on fields in IPv6 headers. Matching on
        layer 4 header fields may also exist in the list.";
   }

   identity mixed-eth-ipv4-ipv6-acl-type {
     base "acl:eth-acl-type";
     base "acl:ipv4-acl-type";
     base "acl:ipv6-acl-type";
     if-feature "mixed-eth-ipv4-ipv6";
     description
       "ACL that contains a mix of entries that
        match on fields in ethernet headers, entries
        that match on fields in IPv4 headers, and entries
        that match on fields in IPv6 headers. Matching on
        layer 4 header fields may also exist in the list.";
   }

   /*
    * Features
    */

   /*
    * Features supported by device
    */
   feature match-on-eth {
     description
       "The device can support matching on ethernet headers.";
   }

   feature match-on-ipv4 {
     description
       "The device can support matching on IPv4 headers.";
```

```
   }

   feature match-on-ipv6 {
     description
       "The device can support matching on IPv6 headers.";
   }

   feature match-on-tcp {
     description
       "The device can support matching on TCP headers.";
   }

   feature match-on-udp {
     description
       "The device can support matching on UDP headers.";
   }

   feature match-on-icmp {
     description
       "The device can support matching on ICMP (v4 and v6) headers.";
   }

   /*
    * Header classifications combinations supported by
    * device
    */
   feature eth {
     if-feature "match-on-eth";
     description
       "Plain Ethernet ACL supported";
   }

   feature ipv4 {
     if-feature "match-on-ipv4";
     description
       "Plain IPv4 ACL supported";
   }

   feature ipv6 {
     if-feature "match-on-ipv6";
     description
       "Plain IPv6 ACL supported";
   }

   feature mixed-eth-ipv4 {
     if-feature "match-on-eth and match-on-ipv4";
     description
       "Ethernet and IPv4 ACL combinations supported";
```

```
   }

   feature mixed-eth-ipv6 {
     if-feature "match-on-eth and match-on-ipv6";
     description
       "Ethernet and IPv6 ACL combinations supported";
   }

   feature mixed-eth-ipv4-ipv6 {
     if-feature "match-on-eth and match-on-ipv4
                 and match-on-ipv6";
     description
       "Ethernet, IPv4 and IPv6 ACL combinations supported.";
   }

   /*
    * Stats Features
    */
   feature interface-stats {
     description
       "ACL counters are available and reported only per interface";
   }

   feature acl-aggregate-stats {
     description
       "ACL counters are aggregated over all interfaces, and reported
        only per ACL entry";
   }

   /*
    * Attachment point features
    */
   feature interface-attachment {
     description
       "ACLs are set on interfaces.";
   }

   /*
    * Typedefs
    */
   typedef acl-type {
     type identityref {
       base acl-base;
     }
     description
       "This type is used to refer to an Access Control List
        (ACL) type";
   }
```

```
   /*
    * Groupings
    */
   grouping acl-counters {
     description
       "Common grouping for ACL counters";

     leaf matched-packets {
       type yang:counter64;
       config false;
       description
         "Count of the number of packets matching the current ACL
          entry.

          An implementation should provide this counter on a
          per-interface per-ACL-entry basis if possible.

          If an implementation only supports ACL counters on a per
          entry basis (i.e., not broken out per interface), then the
          value should be equal to the aggregate count across all
          interfaces.

          An implementation that provides counters on a per entry per
          interface basis is not required to also provide an aggregate
          count, e.g., per entry -- the user is expected to be able
          implement the required aggregation if such a count is
          needed.";
     }

     leaf matched-octets {
       type yang:counter64;
       config false;
       description
         "Count of the number of octets (bytes) matching the current
          ACL entry.

          An implementation should provide this counter on a
          per-interface per-ACL-entry if possible.

          If an implementation only supports ACL counters per entry
          (i.e., not broken out per interface), then the value
          should be equal to the aggregate count across all interfaces.

          An implementation that provides counters per entry per
          interface is not required to also provide an aggregate count,
          e.g., per entry -- the user is expected to be able implement
          the required aggregation if such a count is needed.";
     }
```

```
   }

   /*
    * Configuration and monitoring data nodes
    */
   container acls {
     description
       "This is a top level container for Access Control Lists.
        It can have one or more acl nodes.";
     list acl {
       key "name";
       description
         "An Access Control List (ACL) is an ordered list of
          Access Control Entries (ACE). Each ACE has a
          list of match criteria and a list of actions.
          Since there are several kinds of Access Control Lists
          implemented with different attributes for
          different vendors, this model accommodates customizing
          Access Control Lists for each kind and, for each vendor.";
       leaf name {
         type string {
           length "1..64";
         }
         description
           "The name of access list. A device MAY restrict the length
            and value of this name, possibly space and special
            characters are not allowed.";
       }
       leaf type {
         type acl-type;
         description
           "Type of access control list. Indicates the primary intended
            type of match criteria (e.g. ethernet, IPv4, IPv6, mixed,
            etc) used in the list instance.";
       }
       container aces {
         description
           "The aces container contains one or more ace nodes.";
         list ace {
           key "name";
           ordered-by user;
           description
             "List of Access Control Entries (ACEs)";
           leaf name {
             type string {
               length "1..64";
             }
             description
```

```
               "A unique name identifying this Access Control
                Entry (ACE).";
           }

           container matches {
             description
               "The rules in this set determine what fields will be
                matched upon before any action is taken on them.
                The rules are selected based on the feature set
                defined by the server and the acl-type defined.
                If no matches are defined in a particular container,
                then any packet will match that container. If no
                matches are specified at all in an ACE, then any
                packet will match the ACE.";

             choice l2 {
               container eth {
                 when "derived-from-or-self(/acls/acl/type, " +
                      "'acl:eth-acl-type')";
                 if-feature match-on-eth;
                 uses pf:acl-eth-header-fields;
                 description
                   "Rule set that matches ethernet headers.";
               }
               description
                 "Match layer 2 headers, for example ethernet
                  header fields.";
             }

             choice l3 {
               container ipv4 {
                 when "derived-from-or-self(/acls/acl/type, " +
                      "'acl:ipv4-acl-type')";
                 if-feature match-on-ipv4;
                 uses pf:acl-ip-header-fields;
                 uses pf:acl-ipv4-header-fields;
                 description
                   "Rule set that matches IPv4 headers.";
               }

               container ipv6 {
                 when "derived-from-or-self(/acls/acl/type, " +
                      "'acl:ipv6-acl-type')";
                 if-feature match-on-ipv6;
                 uses pf:acl-ip-header-fields;
                 uses pf:acl-ipv6-header-fields;
                 description
                   "Rule set that matches IPv6 headers.";
```

```
              }
              description
                "Choice of either ipv4 or ipv6 headers";
            }

            choice l4 {
              container tcp {
                if-feature match-on-tcp;
                uses pf:acl-tcp-header-fields;
                container source-port {
                  choice source-port {
                    case range-or-operator {
                      uses pf:port-range-or-operator;
                      description
                        "Source port definition from range or
                         operator.";
                    }
                    description
                      "Choice of source port definition using
                       range/operator or a choice to support future
                       'case' statements, such as one enabling a
                       group of source ports to be referenced.";
                  }
                  description
                    "Source port definition.";
                }
                container destination-port {
                  choice destination-port {
                    case range-or-operator {
                      uses pf:port-range-or-operator;
                      description
                        "Destination port definition from range or
                         operator.";
                    }
                    description
                      "Choice of destination port definition using
                       range/operator or a choice to support future
                       'case' statements, such as one enabling a
                       group of destination ports to be referenced.";
                  }
                  description
                    "Destination port definition.";
                }
                description
                  "Rule set that matches TCP headers.";
              }

              container udp {
```

```
                  if-feature match-on-udp;
                  uses pf:acl-udp-header-fields;
                  container source-port {
                    choice source-port {
                      case range-or-operator {
                        uses pf:port-range-or-operator;
                        description
                          "Source port definition from range or
                           operator.";
                      }
                      description
                        "Choice of source port definition using
                         range/operator or a choice to support future
                         'case' statements, such as one enabling a
                         group of source ports to be referenced.";
                    }
                    description
                      "Source port definition.";
                  }
                  container destination-port {
                    choice destination-port {
                      case range-or-operator {
                        uses pf:port-range-or-operator;
                        description
                          "Destination port definition from range or
                           operator.";
                      }
                      description
                        "Choice of destination port definition using
                         range/operator or a choice to support future
                         'case' statements, such as one enabling a
                         group of destination ports to be referenced.";
                    }
                    description
                      "Destination port definition.";
                  }
                  description
                    "Rule set that matches UDP headers.";
                }

                container icmp {
                  if-feature match-on-icmp;
                  uses pf:acl-icmp-header-fields;
                  description
                    "Rule set that matches ICMP headers.";
                }
                description
                  "Choice of TCP, UDP or ICMP headers.";
```

```
            }

            leaf egress-interface {
              type if:interface-ref;
              description
                "Egress interface. This should not be used if this ACL
                 is attached as an egress ACL (or the value should
                 equal the interface to which the ACL is attached).";
            }

            leaf ingress-interface {
              type if:interface-ref;
              description
                "Ingress interface. This should not be used if this ACL
                 is attached as an ingress ACL (or the value should
                 equal the interface to which the ACL is attached)";
            }
          }

          container actions {
            description
              "Definitions of action for this ace entry";
            leaf forwarding {
              type identityref {
                base forwarding-action;
              }
              mandatory true;
              description
                "Specifies the forwarding action per ace entry";
            }

            leaf logging {
              type identityref {
                base log-action;
              }
              default log-none;
              description
                "Specifies the log action and destination for
                 matched packets. Default value is not to log the
                 packet.";
            }
          }
          container statistics {
            if-feature "acl-aggregate-stats";
            config false;
            description
              "Statistics gathered across all attachment points for the
               given ACL.";
```

```
             uses acl-counters;
           }
         }
       }
     }
     container attachment-points {
       description
         "Enclosing container for the list of
          attachment-points on which ACLs are set";

       /*
        * Groupings
        */
       grouping interface-acl {
         description
           "Grouping for per-interface ingress ACL data";

         container acl-sets {
           description
             "Enclosing container the list of ingress ACLs on the
              interface";

           list acl-set {
             key "name";
             ordered-by user;
             description
               "List of ingress ACLs on the interface";

             leaf name {
               type leafref {
                 path "/acls/acl/name";
               }
               description
                 "Reference to the ACL name applied on ingress";
             }

             list ace-statistics {
               if-feature "interface-stats";
               key "name";
               config false;
               description
                 "List of Access Control Entries (ACEs)";
               leaf name {
                 type leafref {
                   path "/acls/acl/aces/ace/name";
                 }
                 description
                   "The ace name";
```

```
              }
              uses acl-counters;
            }
          }
        }
      }

      list interface {
        if-feature interface-attachment;
        key "interface-id";
        description
          "List of interfaces on which ACLs are set";

        leaf interface-id {
          type if:interface-ref;
          description
            "Reference to the interface id list key";
        }

        container ingress {
          uses interface-acl;
          description
            "The ACLs applied to ingress interface";
        }
        container egress {
          uses interface-acl;
          description
            "The ACLs applied to egress interface";
        }
      }
    }
  }
}

<CODE ENDS>
```

4.2. IETF Packet Fields module

The packet fields module defines the necessary groups for matching on fields in the packet including ethernet, ipv4, ipv6, and transport layer fields. The "type" node determines which of these fields get included for any given ACL with the exception of TCP, UDP and ICMP header fields. Those fields can be used in conjunction with any of the above layer 2 or layer 3 fields.

Since the number of match criteria are very large, the base draft does not include these directly but references them by 'uses' statement to keep the base module simple. In case more match conditions are needed, those can be added by augmenting choices within container "matches" in ietf-access-control-list.yang model.

This module imports definitions from Common YANG Data Types [RFC6991] and references IP [RFC0791], ICMP [RFC0792], TCP [RFC0793], Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers [RFC2474], The Addition of Explicit Congestion Notification (ECN) to IP [RFC3168], , IPv6 Scoped Address Architecture [RFC4007], IPv6 Addressing Architecture [RFC4291], A Recommendation for IPv6 Address Text Representation [RFC5952], IPv6 [RFC8200].

```
<CODE BEGINS> file "ietf-packet-fields@2018-11-06.yang"

module ietf-packet-fields {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-packet-fields";
  prefix packet-fields;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991 - Common YANG Data Types.";
  }

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991 - Common YANG Data Types.";
  }

  import ietf-ethertypes {
    prefix eth;
    reference
      "RFC XXXX - Network ACL YANG Model.";
  }

  organization
    "IETF NETMOD (Network Modeling Language) Working
     Group";

  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
     WG List: netmod@ietf.org

     Editor: Mahesh Jethanandani
             mjethanandani@gmail.com
     Editor: Lisa Huang
             lyihuang16@gmail.com
```

```
     Editor: Sonal Agarwal
             sagarwal12@gmail.com
     Editor: Dana Blair
             dblair@cisco.com";

  description
    "This YANG module defines groupings that are used by
    ietf-access-control-list YANG module. Their usage is not
    limited to ietf-access-control-list and can be
    used anywhere as applicable.

    Copyright (c) 2018 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).

    This version of this YANG module is part of RFC XXXX; see
    the RFC itself for full legal notices.";

  revision 2018-11-06 {
    description
      "Initial version.";
    reference
      "RFC XXX: Network Access Control List (ACL) YANG Data Model.";
  }

  /*
   * Typedefs
   */
  typedef operator {
    type enumeration {
      enum lte {
        description
          "Less than or equal.";
      }
      enum gte {
        description
          "Greater than or equal.";
      }
      enum eq {
        description
          "Equal to.";
      }
      enum neq {
```

```
        description
          "Not equal to.";
      }
    }
    description
      "The source and destination port range definitions
       can be further qualified using an operator. An
       operator is needed only if lower-port is specified
       and upper-port is not specified. The operator
       therefore further qualifies lower-port only.";
  }

  /*
   * Groupings
   */
  grouping port-range-or-operator {
    choice port-range-or-operator {
      case range {
        leaf lower-port {
          type inet:port-number;
          must ". <= ../upper-port" {
            error-message
              "The lower-port must be less than or equal to
               upper-port.";
          }
          mandatory true;
          description
            "Lower boundry for a port.";
        }
        leaf upper-port {
          type inet:port-number;
          mandatory true;
          description
            "Upper boundry for port.";
        }
      }
      case operator {
        leaf operator {
          type operator;
          default eq;
          description
            "Operator to be applied on the port below.";
        }
        leaf port {
          type inet:port-number;
          mandatory true;
          description
            "Port number along with operator on which to
```

```
             match.";
        }
      }
      description
        "Choice of specifying a port range or a single
         port along with an operator.";
    }
    description
      "Grouping for port definitions in the form of a
       choice statement.";
  }

  grouping acl-ip-header-fields {
    description
      "IP header fields common to ipv4 and ipv6";
    reference
      "RFC 791: Internet Protocol.";

    leaf dscp {
      type inet:dscp;
      description
        "Differentiated Services Code Point.";
      reference
        "RFC 2474: Definition of Differentiated services field
         (DS field) in the IPv4 and IPv6 headers.";
    }

    leaf ecn {
      type uint8 {
        range 0..3;
      }
      description
        "Explicit Congestion Notification.";
      reference
        "RFC 3168: Explicit Congestion Notification.";
    }

    leaf length {
      type uint16;
      description
        "In IPv4 header field, this field is known as the Total Length.
         Total Length is the length of the datagram, measured in octets,
         including internet header and data.

         In IPv6 header field, this field is known as the Payload
         Length, the length of the IPv6 payload, i.e. the rest of
         the packet following the IPv6 header, in octets.";
      reference
```

```
          "RFC 791: Internet Protocol,
           RFC 8200: Internet Protocol, Version 6 (IPv6) Specification.";
     }

     leaf ttl {
       type uint8;
       description
         "This field indicates the maximum time the datagram is allowed
          to remain in the internet system.  If this field contains the
          value zero, then the datagram must be dropped.

          In IPv6, this field is known as the Hop Limit.";
       reference
         "RFC 791: Internet Protocol,
          RFC 8200: Internet Protocol, Version 6 (IPv6) Specification.";
     }

     leaf protocol {
       type uint8;
       description
         "Internet Protocol number. Refers to the protocol of the
          payload. In IPv6, this field is known as 'next-header,
          and if extension headers are present, the protocol is
          present in the 'upper-layer' header.";
       reference
         "RFC 791: Internet Protocol,
          RFC 8200: Internet Protocol, Version 6 (IPv6) Specification.";
     }
   }

   grouping acl-ipv4-header-fields {
     description
       "Fields in IPv4 header.";

     leaf ihl {
       type uint8 {
         range "5..60";
       }
       description
         "An IPv4 header field, the Internet Header Length (IHL) is
          the length of the internet header in 32 bit words, and
          thus points to the beginning of the data. Note that the
          minimum value for a correct header is 5.";
     }

     leaf flags {
       type bits {
         bit reserved {
```

```
          position 0;
          description
            "Reserved. Must be zero.";
        }
        bit fragment {
          position 1;
          description
            "Setting value to 0 indicates may fragment, while setting
             the value to 1 indicates do not fragment.";
        }
        bit more {
          position 2;
          description
            "Setting the value to 0 indicates this is the last fragment,
             and setting the value to 1 indicates more fragments are
             coming.";
        }
      }
      description
        "Bit definitions for the flags field in IPv4 header.";
    }

    leaf offset {
      type uint16 {
        range "20..65535";
      }
      description
        "The fragment offset is measured in units of 8 octets (64 bits).
         The first fragment has offset zero. The length is 13 bits";
    }

    leaf identification {
      type uint16;
      description
        "An identifying value assigned by the sender to aid in
         assembling the fragments of a datagram.";
    }

    choice destination-network {
      case destination-ipv4-network {
        leaf destination-ipv4-network {
          type inet:ipv4-prefix;
          description
            "Destination IPv4 address prefix.";
        }
      }
      description
        "Choice of specifying a destination IPv4 address or
```

```
         referring to a group of IPv4 destination addresses.";
    }
    choice source-network {
      case source-ipv4-network {
        leaf source-ipv4-network {
          type inet:ipv4-prefix;
          description
            "Source IPv4 address prefix.";
        }
      }
      description
        "Choice of specifying a source IPv4 address or
         referring to a group of IPv4 source addresses.";
    }
  }

  grouping acl-ipv6-header-fields {
    description
      "Fields in IPv6 header";

    choice destination-network {
      case destination-ipv6-network {
        leaf destination-ipv6-network {
          type inet:ipv6-prefix;
          description
            "Destination IPv6 address prefix.";
        }
      }
      description
        "Choice of specifying a destination IPv6 address
         or referring to a group of IPv6 destination
         addresses.";
    }

    choice source-network {
      case source-ipv6-network {
        leaf source-ipv6-network {
          type inet:ipv6-prefix;
            description
              "Source IPv6 address prefix.";
        }
      }
      description
        "Choice of specifying a source IPv6 address or
         referring to a group of IPv6 source addresses.";
    }

    leaf flow-label {
```

```
      type inet:ipv6-flow-label;
      description
        "IPv6 Flow label.";
    }
    reference
      "RFC 4291: IP Version 6 Addressing Architecture
       RFC 4007: IPv6 Scoped Address Architecture
       RFC 5952: A Recommendation for IPv6 Address Text
                 Representation";
  }

  grouping acl-eth-header-fields {
    description
      "Fields in Ethernet header.";

    leaf destination-mac-address {
      type yang:mac-address;
      description
        "Destination IEEE 802 MAC address.";
    }
    leaf destination-mac-address-mask {
      type yang:mac-address;
      description
        "Destination IEEE 802 MAC address mask.";
    }
    leaf source-mac-address {
      type yang:mac-address;
      description
        "Source IEEE 802 MAC address.";
    }
    leaf source-mac-address-mask {
      type yang:mac-address;
      description
        "Source IEEE 802 MAC address mask.";
    }
    leaf ethertype {
      type eth:ethertype;
      description
        "The Ethernet Type (or Length) value represented
         in the canonical order defined by IEEE 802.
         The canonical representation uses lowercase
         characters.";
      reference
        "IEEE 802-2014 Clause 9.2";
    }
    reference
      "IEEE 802: IEEE Standard for Local and Metropolitan
       Area Networks: Overview and Architecture.";
```

```
  }

  grouping acl-tcp-header-fields {
    description
      "Collection of TCP header fields that can be used to
       setup a match filter.";

    leaf sequence-number {
      type uint32;
      description
        "Sequence number that appears in the packet.";
    }

    leaf acknowledgement-number {
      type uint32;
      description
        "The acknowledgement number that appears in the
         packet.";
    }

    leaf data-offset {
      type uint8 {
        range "5..15";
      }
      description
        "Specifies the size of the TCP header in 32-bit
         words. The minimum size header is 5 words and
         the maximum is 15 words thus giving the minimum
         size of 20 bytes and maximum of 60 bytes,
         allowing for up to 40 bytes of options in the
         header.";
    }

    leaf reserved {
      type uint8;
      description
        "Reserved for future use.";
    }

    leaf flags {
      type bits {
        bit cwr {
          position 1;
          description
            "Congestion Window Reduced (CWR) flag is set by
             the sending host to indicate that it received
             a TCP segment with the ECE flag set and had
             responded in congestion control mechanism.";
```

```
          reference
            "RFC 3168: The Addition of Explicit Congestion
                       Notification (ECN) to IP.";
        }
        bit ece {
          position 2;
          description
            "ECN-Echo has a dual role, depending on the value
             of the SYN flag. It indicates:
             If the SYN flag is set (1), that the TCP peer is ECN
             capable. If the SYN flag is clear (0), that a packet
             with Congestion Experienced flag set (ECN=11) in IP
             header was received during normal transmission
             (added to header by RFC 3168). This serves as an
             indication of network congestion (or impending
             congestion) to the TCP sender.";
          reference
            "RFC 3168: The Addition of Explicit Congestion
                       Notification (ECN) to IP.";
        }
        bit urg {
          position 3;
          description
            "Indicates that the Urgent pointer field is significant.";
        }
        bit ack {
          position 4;
          description
            "Indicates that the Acknowledgment field is significant.
             All packets after the initial SYN packet sent by the
             client should have this flag set.";
        }
        bit psh {
          position 5;
          description
            "Push function. Asks to push the buffered data to the
             receiving application.";
        }
        bit rst {
          position 6;
          description
            "Reset the connection.";
        }
        bit syn {
          position 7;
          description
            "Synchronize sequence numbers. Only the first packet
             sent from each end should have this flag set. Some
```

```
             other flags and fields change meaning based on this
             flag, and some are only valid for when it is set,
             and others when it is clear.";
        }
        bit fin {
          position 8;
          description
            "Last package from sender.";
        }
      }
      description
        "Also known as Control Bits. Contains 9 1-bit flags.";
      reference
        "RFC 793: Transmission Control Protocol (TCP).";
    }

    leaf window-size {
      type uint16;
      units "bytes";
      description
        "The size of the receive window, which specifies
         the number of window size units beyond the segment
         identified by the sequence number in the acknowledgment
         field that the sender of this segment is currently
         willing to receive.";
    }

    leaf urgent-pointer {
      type uint16;
      description
        "This field is an offset from the sequence number
         indicating the last urgent data byte.";
    }

    leaf options {
      type binary {
        length "1..40";
      }
      description
        "The length of this field is determined by the
         data offset field. Options have up to three
         fields: Option-Kind (1 byte), Option-Length
         (1 byte), Option-Data (variable). The Option-Kind
         field indicates the type of option, and is the
         only field that is not optional. Depending on
         what kind of option we are dealing with,
         the next two fields may be set: the Option-Length
         field indicates the total length of the option,
```

```
         and the Option-Data field contains the value of
         the option, if applicable.";
    }
  }

  grouping acl-udp-header-fields {
    description
      "Collection of UDP header fields that can be used
       to setup a match filter.";

    leaf length {
      type uint16;
      description
        "A field that specifies the length in bytes of
         the UDP header and UDP data. The minimum
         length is 8 bytes because that is the length of
         the header. The field size sets a theoretical
         limit of 65,535 bytes (8 byte header + 65,527
         bytes of data) for a UDP datagram. However the
         actual limit for the data length, which is
         imposed by the underlying IPv4 protocol, is
         65,507 bytes (65,535 minus 8 byte UDP header
         minus 20 byte IP header).

         In IPv6 jumbograms it is possible to have
         UDP packets of size greater than 65,535 bytes.
         RFC 2675 specifies that the length field is set
         to zero if the length of the UDP header plus
         UDP data is greater than 65,535.";
    }
  }

  grouping acl-icmp-header-fields {
    description
      "Collection of ICMP header fields that can be
       used to setup a match filter.";

    leaf type {
      type uint8;
      description
        "Also known as Control messages.";
      reference
        "RFC 792: Internet Control Message Protocol (ICMP),
         RFC 4443: Internet Control Message Protocol (ICMPv6)
                   for Internet Protocol Version 6 (IPv6)
                   Specifciation.";
    }
```

```
    leaf code {
      type uint8;
      description
        "ICMP subtype. Also known as Control messages.";
      reference
        "RFC 792: Internet Control Message Protocol (ICMP),
         RFC 4443: Internet Control Message Protocol (ICMPv6)
                   for Internet Protocol Version 6 (IPv6)
                   Specifciation.";
    }

    leaf rest-of-header {
      type binary;
      description
        "Unbounded in length, the contents vary based on the
         ICMP type and code. Also referred to as 'Message Body'
         in ICMPv6.";
      reference
        "RFC 792: Internet Control Message Protocol (ICMP),
         RFC 4443: Internet Control Message Protocol (ICMPv6)
                   for Internet Protocol Version 6 (IPv6)
                   Specifciation.";
    }
  }
}

<CODE ENDS>
```

4.3. ACL Examples

Requirement: Deny tcp traffic from 192.0.2.0/24, destined to 198.51.100.0/24.

Here is the acl configuration xml for this Access Control List:

[note: ’\’ line wrapping for formatting only]

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-ipv4-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <ipv4>
              <protocol>6</protocol>
              <destination-ipv4-network>198.51.100.0/24</destination\
-ipv4-network>
              <source-ipv4-network>192.0.2.0/24</source-ipv4-network\
>
            </ipv4>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```

The acl and aces can be described in CLI as the following:

```
      acl ipv4 sample-ipv4-acl
      deny tcp 192.0.2.0/24 198.51.100.0/24
```

Requirement: Accept all DNS traffic destined for 2001:db8::/32 on port 53.

[note: '\' line wrapping for formatting only]


```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>allow-dns-packets</name>
      <type>ipv6-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <ipv6>
              <destination-ipv6-network>2001:db8::/32</destination-i\
pv6-network>
            </ipv6>
            <udp>
              <destination-port>
                <operator>eq</operator>
                <port>53</port>
              </destination-port>
            </udp>
          </matches>
          <actions>
            <forwarding>accept</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```


4.4. Port Range Usage and Other Examples

When a lower-port and an upper-port are both present, it represents a range between lower-port and upper-port with both the lower-port and upper-port included. When only a port is present, it represents a port, with the operator specifying the range.

The following XML example represents a configuration where TCP traffic from source ports 16384, 16385, 16386, and 16387 is dropped.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-port-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <tcp>
              <source-port>
                <lower-port>16384</lower-port>
                <upper-port>16387</upper-port>
              </source-port>
            </tcp>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```


The following XML example represents a configuration where all IPv4 ICMP echo requests are dropped.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-icmp-acl</name>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <ipv4>
              <protocol>1</protocol>
            </ipv4>
            <icmp>
              <type>8</type>
              <code>0</code>
            </icmp>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```


The following XML example represents a configuration of a single port, port 21 that accepts TCP traffic.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-ipv4-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <tcp>
              <destination-port>
                <operator>eq</operator>
                <port>21</port>
              </destination-port>
            </tcp>
          </matches>
          <actions>
            <forwarding>accept</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```


The following XML example represents a configuration specifying all ports that are not equal to 21, that will drop TCP packets destined for those ports.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <acls
    xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
    <acl>
      <name>sample-ipv4-acl</name>
      <type>ipv4-acl-type</type>
      <aces>
        <ace>
          <name>rule1</name>
          <matches>
            <tcp>
              <destination-port>
                <operator>neq</operator>
                <port>21</port>
              </destination-port>
            </tcp>
          </matches>
          <actions>
            <forwarding>drop</forwarding>
          </actions>
        </ace>
      </aces>
    </acl>
  </acls>
</config>
```


5. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocol such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF Access Control Model (NACM [RFC8341]) provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which 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:

/acls/acl/aces: This list specifies all the configured access control entries on the device. Unauthorized write access to this list can allow intruders to modify the entries so as to permit traffic that should not be permitted, or deny traffic that should be permitted. The former may result in a DoS attack, or compromise the device. The latter may result in a DoS attack. The impact of an unauthorized read access of the list will allow the attacker to determine which rules are in effect, to better craft an attack.

/acls/acl/aces/ace/actions/logging: This node specifies ability to log packets that match this ace entry. Unauthorized write access to this node can allow intruders to enable logging on one or many ace entries, overwhelming the server in the process. Unauthorized read access of this node can allow intruders to access logging information, which could be used to craft an attack the server.

6. IANA Considerations

This document registers three URIs and three YANG modules.

6.1. URI Registration

This document registers three URIs in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-access-control-list
URI: urn:ietf:params:xml:ns:yang:ietf-packet-fields
URI: urn:ietf:params:xml:ns:yang:ietf-ethertypes

Registrant Contact: The IESG.

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

6.2. YANG Module Name Registration

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

```
name: ietf-access-control-list
namespace: urn:ietf:params:xml:ns:yang:ietf-access-control-list
prefix: acl
reference: RFC XXXX

name: ietf-packet-fields
namespace: urn:ietf:params:xml:ns:yang:ietf-packet-fields
prefix: packet-fields
reference: RFC XXXX

name: ietf-ethertypes
namespace: urn:ietf:params:xml:ns:yang:ietf-ethertypes
prefix: ethertypes
reference: RFC XXXX
```

7. Acknowledgements

Alex Clemm, Andy Bierman and Lisa Huang started it by sketching out an initial IETF draft in several past IETF meetings. That draft included an ACL YANG model structure and a rich set of match filters, and acknowledged contributions by Louis Fourie, Dana Blair, Tula Kraiser, Patrick Gili, George Serpa, Martin Bjorklund, Kent Watsen, and Phil Shafer. Many people have reviewed the various earlier drafts that made the draft went into IETF charter.

Dean Bogdanovic, Kiran Agrahara Sreenivasa, Lisa Huang, and Dana Blair each evaluated the YANG model in previous drafts separately, and then worked together to created a ACL draft that was supported by different vendors. That draft removed vendor specific features, and gave examples to allow vendors to extend in their own proprietary ACL. The earlier draft was superseded with this updated draft and received more participation from many vendors.

Authors would like to thank Jason Sterne, Lada Lhotka, Juergen Schoenwalder, David Bannister, Jeff Haas, Kristian Larsson and Einar Nilsen-Nygaard for their review of and suggestions to the draft.

8. References

8.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <https://www.rfc-editor.org/info/rfc792>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[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>.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, DOI 10.17487/RFC4007, March 2005, <https://www.rfc-editor.org/info/rfc4007>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[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>.

[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>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

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

8.2. Informative References

[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>.

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

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

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[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>.

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

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Appendix A. Extending ACL model examples

A.1. A company proprietary module example

Module "example-newco-acl" is an example of company proprietary model that augments "ietf-acl" module. It shows how to use 'augment' with an XPath expression to add additional match criteria, actions, and default actions for when no ACE matches are found. All these are company proprietary extensions or system feature extensions. "example-newco-acl" is just an example and it is expected that vendors will create their own proprietary models.

```
module example-newco-acl {

  yang-version 1.1;

  namespace "http://example.com/ns/example-newco-acl";

  prefix example-newco-acl;

  import ietf-access-control-list {
    prefix "acl";
  }

  organization
    "Newco model group.";

  contact
    "abc@newco.com";
  description
    "This YANG module augments IETF ACL Yang.";

  revision 2018-11-06 {
    description
      "Creating NewCo proprietary extensions to ietf-acl model";

    reference
      "RFC XXXX: Network Access Control List (ACL)
       YANG Data  Model";
  }

  augment "/acl:acls/acl:acl/" +
          "acl:aces/acl:ace/" +
          "acl:matches" {
```

```
       description "Newco proprietary simple filter matches";
       choice protocol-payload-choice {
         description "Newco proprietary payload match condition";
         list protocol-payload {
           key value-keyword;
           ordered-by user;
           description "Match protocol payload";
           uses match-simple-payload-protocol-value;
         }
       }

       choice metadata {
         description "Newco proprietary interface match condition";
         leaf packet-length {
           type uint16;
           description "Match on packet length";
         }
       }
     }

     augment "/acl:acls/acl:acl/" +
             "acl:aces/acl:ace/" +
             "acl:actions" {
       description "Newco proprietary simple filter actions";
       choice action {
         description "";
         case count {
           description "Count the packet in the named counter";
           leaf count {
             type uint32;
             description "Count";
           }
         }
         case policer {
           description "Name of policer to use to rate-limit traffic";
           leaf policer {
             type string;
             description "Name of the policer";
           }
         }
         case hiearchical-policer {
           leaf hierarchitacl-policer {
             type string;
             description
               "Name of the hierarchical policer.";
           }
           description
             "Name of hierarchical policer to use to
```

```
              rate-limit traffic";
        }
      }
    }

    augment "/acl:acls/acl:acl" +
            "/acl:aces/acl:ace/" +
            "acl:actions" {
      leaf default-action {
        type identityref {
          base acl:forwarding-action;
        }
        default acl:drop;
        description
          "Actions that occur if no ace is matched.";
      }
      description
        "Newco proprietary default action";
    }

    grouping match-simple-payload-protocol-value {
      description "Newco proprietary payload";
      leaf value-keyword {
        type enumeration {
          enum icmp {
            description "Internet Control Message Protocol";
          }
          enum icmp6 {
            description
              "Internet Control Message Protocol
               Version 6";
          }
          enum range {
            description "Range of values";
          }
        }
        description "(null)";
      }
    }
  }
```

The following figure is the tree diagram of example-newco-acl. In this example, /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace/ ietf-acl:matches are augmented with two new choices, protocol-payload-choice and metadata. The protocol-payload-choice uses a grouping with an enumeration of all supported protocol values. Metadata matches apply to fields associated with the packet but not in the packet header such as overall packet length. In another example, /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace/ietf-acl:actions are augmented with a new choice of actions.

```
module: example-newco-acl
  augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches:
    +--rw (protocol-payload-choice)?
    |  +--:(protocol-payload)
    |     +--rw protocol-payload* [value-keyword]
    |        +--rw value-keyword    enumeration
    +--rw (metadata)?
       +--:(packet-length)
          +--rw packet-length?      uint16
  augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:actions:
    +--rw (action)?
       +--:(count)
       |  +--rw count?                    uint32
       +--:(policer)
       |  +--rw policer?                  string
       +--:(hiearchical-policer)
          +--rw hierarchitacl-policer?   string
  augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:actions:
    +--rw default-action?   identityref
```

A.2. Linux nftables

As Linux platform is becoming more popular as networking platform, the Linux data model is changing. Previously ACLs in Linux were highly protocol specific and different utilities were used (iptables, ip6tables, arptables, ebtables), so each one had separate data model. Recently, this has changed and a single utility, nftables, has been developed. With a single application, it has a single data model for filewall filters and it follows very similarly to the ietf-access-control list module proposed in this draft. The nftables support input and output ACEs and each ACE can be defined with match and action.

The example in Section 4.3 can be configured using nftable tool as below.

```
nft add table ip filter
nft add chain filter input
nft add rule ip filter input ip protocol tcp ip saddr \
    192.0.2.1/24 drop
```

The configuration entries added in nftable would be.

```
table ip filter {
  chain input {
    ip protocol tcp ip saddr 192.0.2.1/24 drop
  }
}
```

We can see that there are many similarities between Linux nftables and IETF ACL YANG data models and its extension models. It should be fairly easy to do translation between ACL YANG model described in this draft and Linux nftables.

A.3. Ethertypes

The ACL module is dependent on the definition of ethertypes. IEEE owns the allocation of those ethertypes. This model is being included here to enable definition of those types till such time that IEEE takes up the task of publication of the model that defines those ethertypes. At that time, this model can be deprecated.

<CODE BEGINS> file "ietf-ethertypes@2018-11-06.yang"

```
module ietf-ethertypes {
  namespace "urn:ietf:params:xml:ns:yang:ietf-ethertypes";
  prefix ethertypes;

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

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

     Editor:   Mahesh Jethanandani
               <mjethanandani@gmail.com>";

  description
    "This module contains the common definitions for the
     Ethertype used by different modules. It is a
     placeholder module, till such time that IEEE
     starts a project to define these Ethertypes
     and publishes a standard.

     At that time this module can be deprecated.";

  revision 2018-11-06 {
    description
      "Initial revision.";
```

```
       reference
         "RFC XXXX: IETF Ethertype YANG Data Module.";
     }

     typedef ethertype {
       type union {
         type uint16;
         type enumeration {
           enum ipv4 {
             value 2048;
             description
               "Internet Protocol version 4 (IPv4) with a
                hex value of 0x0800.";
             reference
               "RFC 791: Internet Protocol.";
           }
           enum arp {
             value 2054;
             description
               "Address Resolution Protocol (ARP) with a
                hex value of 0x0806.";
             reference
               "RFC 826: An Ethernet Address Resolution Protocol.";
           }
           enum wlan {
             value 2114;
             description
               "Wake-on-LAN. Hex value of 0x0842.";
           }
           enum trill {
             value 8947;
             description
               "Transparent Interconnection of Lots of Links.
                Hex value of 0x22F3.";
             reference
               "RFC 6325: Routing Bridges (RBridges): Base Protocol
                Specification.";
           }
           enum srp {
             value 8938;
             description
               "Stream Reservation Protocol. Hex value of
                0x22EA.";
             reference
               "IEEE 801.1Q-2011.";
           }
           enum decnet {
             value 24579;
```

```
            description
              "DECnet Phase IV. Hex value of 0x6003.";
          }
          enum rarp {
            value 32821;
            description
              "Reverse Address Resolution Protocol.
               Hex value 0x8035.";
            reference
              "RFC 903. A Reverse Address Resolution Protocol.";
          }
          enum appletalk {
            value 32923;
            description
              "Appletalk (Ethertalk). Hex value 0x809B.";
          }
          enum aarp {
            value 33011;
            description
              "Appletalk Address Resolution Protocol. Hex value
               of 0x80F3.";
          }
          enum vlan {
            value 33024;
            description
              "VLAN-tagged frame (802.1Q) and Shortest Path
               Bridging IEEE 802.1aq with NNI compatibility.
               Hex value 0x8100.";
            reference
              "802.1Q.";
          }
          enum ipx {
            value 33079;
            description
              "Internetwork Packet Exchange (IPX). Hex value
               of 0x8137.";
          }
          enum qnx {
            value 33284;
            description
              "QNX Qnet. Hex value of 0x8204.";
          }
          enum ipv6 {
            value 34525;
            description
              "Internet Protocol Version 6 (IPv6). Hex value
               of 0x86DD.";
            reference
```

```
                "RFC 8200: Internet Protocol, Version 6 (IPv6)
                           Specification
                 RFC 8201: Path MTU Discovery for IPv6.";
            }
            enum efc {
              value 34824;
              description
                "Ethernet flow control using pause frames.
                 Hex value of 0x8808";
              reference
                "IEEE Std. 802.1Qbb.";
            }
            enum esp {
              value 34825;
              description
                "Ethernet Slow Protocol. Hex value of 0x8809.";
              reference
                "IEEE Std. 802.3-2015";
            }
            enum cobranet {
              value 34841;
              description
                "CobraNet. Hex value of 0x8819";
            }
            enum mpls-unicast {
              value 34887;
              description
                "MultiProtocol Label Switch (MPLS) unicast traffic.
                 Hex value of 0x8847.";
              reference
                "RFC 3031: Multiprotocol Label Switching Architecture.";
            }
            enum mpls-multicast {
              value 34888;
              description
                "MultiProtocol Label Switch (MPLS) multicast traffic.
                 Hex value of 0x8848.";
              reference
                "RFC 3031: Multiprotocol Label Switching Architecture.";
            }
            enum pppoe-discovery {
              value 34915;
              description
                "Point-to-Point Protocol over Ethernet. Used during
                 the discovery process. Hex value of 0x8863.";
              reference
                "RFC 2516: A method for Transmitting PPP over Ethernet
                           PPPoE.";
```

```
          }
          enum pppoe-session {
            value 34916;
            description
              "Point-to-Point Protocol over Ethernet. Used during
               session stage. Hex value of 0x8864.";
            reference
              "RFC 2516: A method for Transmitting PPP over Ethernet
                         PPPoE.";
          }
          enum intel-ans {
            value 34925;
            description
              "Intel Advanced Networking Services. Hex value of
               0x886D.";
          }
          enum jumbo-frames {
            value 34928;
            description
              "Jumbo frames or Ethernet frames with more than
               1500 bytes of payload, upto 9000 bytes.";
          }
          enum homeplug {
            value 34939;
            description
              "Family name for the various power line
               communications. Hex value of 0x887B.";
          }
          enum eap {
            value 34958;
            description
              "Ethernet Access Protocol (EAP) over LAN. Hex value
               of 0x888E.";
            reference
              "IEEE 802.1X";
          }
          enum profinet {
            value 34962;
            description
              "PROcess FIeld Net (PROFINET). Hex value of 0x8892.";
          }
          enum hyperscsi {
            value 34970;
            description
              "SCSI over Ethernet. Hex value of 0x889A";
          }
          enum aoe {
            value 34978;
```

```
            description
              "Advanced Technology Advancement (ATA) over Ethernet.
               Hex value of 0x88A2.";
          }
          enum ethercat {
            value 34980;
            description
              "Ethernet for Control Automation Technology (EtherCAT).
               Hex value of 0x88A4.";
          }
          enum provider-bridging {
            value 34984;
            description
              "Provider Bridging (802.1ad) and Shortest Path Bridging
               (801.1aq). Hex value of 0x88A8.";
            reference
              "IEEE 802.1ad, IEEE 802.1aq).";
          }
          enum ethernet-powerlink {
            value 34987;
            description
              "Ethernet Powerlink. Hex value of 0x88AB.";
          }
          enum goose {
            value 35000;
            description
              "Generic Object Oriented Substation Event (GOOSE).
               Hex value of 0x88B8.";
            reference
              "IEC/ISO 8802-2 and 8802-3.";
          }
          enum gse {
            value 35001;
            description
              "Generic Substation Events. Hex value of 88B9.";
            reference
              "IEC 61850.";
          }
          enum sv {
            value 35002;
            description
              "Sampled Value Transmission. Hex value of 0x88BA.";
            reference
              "IEC 61850.";
          }
          enum lldp {
            value 35020;
            description
```

```
              "Link Layer Discovery Protocol (LLDP). Hex value of
               0x88CC.";
            reference
              "IEEE 802.1AB.";
          }
          enum sercos {
            value 35021;
            description
              "Sercos Interface. Hex value of 0x88CD.";
          }
          enum wsmp {
            value 35036;
            description
              "WAVE Short Message Protocl (WSMP). Hex value of
               0x88DC.";
          }
          enum homeplug-av-mme {
            value 35041;
            description
              "HomePlug AV MME. Hex value of 88E1.";
          }
          enum mrp {
            value 35043;
            description
              "Media Redundancy Protocol (MRP). Hex value of
               0x88E3.";
            reference
              "IEC62439-2.";
          }
          enum macsec {
            value 35045;
            description
              "MAC Security. Hex value of 0x88E5.";
            reference
              "IEEE 802.1AE.";
          }
          enum pbb {
            value 35047;
            description
              "Provider Backbone Bridges (PBB). Hex value of
               0x88E7.";
            reference
              "IEEE 802.1ah.";
          }
          enum cfm {
            value 35074;
            description
              "Connectivity Fault Management (CFM). Hex value of
```

```
               0x8902.";
            reference
              "IEEE 802.1ag.";
          }
          enum fcoe {
            value 35078;
            description
              "Fiber Channel over Ethernet (FCoE). Hex value of
               0x8906.";
            reference
              "T11 FC-BB-5.";
          }
          enum fcoe-ip {
            value 35092;
            description
              "FCoE Initialization Protocol. Hex value of 0x8914.";
          }
          enum roce {
            value 35093;
            description
              "RDMA over Converged Ethernet (RoCE). Hex value of
               0x8915.";
          }
          enum tte {
            value 35101;
            description
              "TTEthernet Protocol Control Frame (TTE). Hex value
               of 0x891D.";
            reference
              "SAE AS6802.";
          }
          enum hsr {
            value 35119;
            description
              "High-availability Seamless Redundancy (HSR). Hex
               value of 0x892F.";
            reference
              "IEC 62439-3:2016.";
          }
        }
      }
      description
        "The uint16 type placeholder is defined to enable
         users to manage their own ethertypes not
         covered by the module. Otherwise the module contains
         enum definitions for the more commonly used ethertypes.";
  }
}
```

<CODE ENDS>

Authors' Addresses

Mahesh Jethanandani
VMware

Email: mjethanandani@gmail.com

Sonal Agarwal
Cisco Systems, Inc.

Email: sagarwal12@gmail.com

Lisa Huang

Email: huangyi_99@yahoo.com

Dana Blair

Email: dana@blairhome.com

Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: May 3, 2018

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems

October 30, 2017

Network Management Datastore Architecture
draft-ietf-netmod-revised-datastores-06

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.

Status of This Memo

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.

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 two different datastores, one for configured values and one for operational state values. 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.

- o 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.

- o schema node: A node in the schema tree. The formal definition is in RFC 7950.

- o 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.

- o 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:

- o client: An entity that can access YANG-defined data on a server, over some network management protocol.

- o server: An entity that provides access to YANG-defined data to a client, over some network management protocol.

- o notification: A server-initiated message indicating that a certain event has been recognized by the server.

- o remote procedure call: An operation that can be invoked by a client on a server.

4. Background

NETCONF [RFC6241] provides the following definitions:

- o 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.

- o 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):

- o datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.

- o 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:

- o 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 this document 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
```

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:

o 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 "inactive"
                            |        // nodes, 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
```

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. The set of datastores include:

- o <running>

- o <candidate>

o <startup>

o <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 complete current configuration on 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>.

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 YANG 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:

- o origin: abstract base identity from which the other origin identities are derived.
- o intended: represents configuration provided by <intended>.
- o dynamic: represents configuration provided by a dynamic configuration datastore.
- o 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.
- o learned: represents configuration that has been learned via protocol interactions with other systems, including protocols such as link-layer negotiations, routing protocols, DHCP, etc.

o 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.

o 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:

o 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.

o 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.

o 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.

o 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.

7. YANG Modules

<CODE BEGINS> file "ietf-datastores@2017-08-17.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) 2017 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

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

        This version of this YANG module is part of RFC XXXX
        (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
        for full legal notices.";

     revision 2017-08-17 {
       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@2017-08-17.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) 2017 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

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

        This version of this YANG module is part of RFC XXXX
        (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
        for full legal notices.";

     revision 2017-08-17 {
       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 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 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 meta-data, hence the "YANG module security guidelines" do not apply.

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:

- o Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- o Andy Bierman, YumaWorks, <andy@yumaworks.com>
- o Marcus Hines, Google, <hines@google.com>
- o Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- o Balazs Lengyel, Ericsson, <balazs.lengyel@ericsson.com>
- o Acee Lindem, Cisco Systems, <acee@cisco.com>
- o Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- o Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- o Tom Petch, Engineering Networks Ltd, <ietfc@btconnect.com>

o Anees Shaikh, Google, <aashaikh@google.com>

o Rob Shakir, Google, <robjs@google.com>

o 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</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:dynamic">bar</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:dynamic">
      <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 ipaddress;
         }
         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>10.1.2.3</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>10.1.2.3</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>10.1.2.3</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

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

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.

o 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.

o 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:

o client: An entity that can access YANG-defined data on a server, over some network management protocol.

o server: An entity that provides access to YANG-defined data to a client, over some network management protocol.

o notification: A server-initiated message indicating that a certain event has been recognized by the server.

o remote procedure call: An operation that can be invoked by a client on a server.

4. Background

NETCONF [RFC6241] provides the following definitions:

o 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.

o 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):

o datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.

o 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:

o 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:

- o Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.

- o 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.

- o Several implementations have proprietary mechanisms that allow clients to store inactive data in <running>. Inactive data is conceptually removed before validation.

- o 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.

- o 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:

- o <running>
- o <candidate>
- o <startup>
- o <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:

- o origin: abstract base identity from which the other origin identities are derived.

- o intended: represents configuration provided by <intended>.

o dynamic: represents configuration provided by a dynamic configuration datastore.

o 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.

o learned: represents configuration that has been learned via protocol interactions with other systems, including protocols such as link-layer negotiations, routing protocols, DHCP, etc.

o 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.

o 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:

- o 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.

- o 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.

- o 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.

- o 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

NETMOD Working Group L. Lhotka
Internet-Draft CZ.NIC
Intended status: Standards Track A. Lindem
Expires: May 4, 2018 Cisco Systems
Y. Qu
Futurewei Technologies, Inc.
October 31, 2017

A YANG Data Model for Routing Management (NDMA Version)
draft-ietf-netmod-rfc8022bis-01

Abstract

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

This version of these YANG modules uses new names for these YANG models. The main difference from the first version is that this version fully conforms to the Network Management Datastore Architecture (NMDA). Consequently, this document obsoletes RFC 8022.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document contains a specification of the following YANG modules:

- o The "ietf-routing" module provides generic components of a routing data model.

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

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

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

This version of these YANG modules uses new names for these YANG models. The main difference from the first version is that this version fully conforms to the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores]. Consequently, this document obsoletes RFC 8022 [RFC8022].

2. Terminology and Notation

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

The following terms are defined in [RFC6241]:

- o client

- o message

o protocol operation

o server

The following terms are defined in [RFC7950]:

o action

o augment

o configuration data

o container

o container with presence

o data model

o data node

o feature

o leaf

o list

o mandatory node

o module

o schema tree

o state data

o RPC (Remote Procedure Call) operation

2.1. Glossary of New Terms

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

direct route: a route to a directly connected network.

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

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

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

2.2. Tree Diagrams

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

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

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

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

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

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

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

2.3. Prefixes in Data Node Names

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

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

Table 1: Prefixes and Corresponding YANG Modules

3. Objectives

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

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

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

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

- o Because device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces, the framework should be flexible enough to facilitate that and accommodate data models with different logic.

4. The Design of the Core Routing Data Model

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

Figure 1 shows abridged views of the hierarchies. See Appendix A for the complete data trees.

```
+--rw routing
   +--rw router-id?                 yang:dotted-quad
   +--ro interfaces
   |  +--ro interface*   if:interface-ref
   +--rw control-plane-protocols
   |  +--rw control-plane-protocol* [type name]
   |     +--rw type             identityref
   |     +--rw name             string
   |     +--rw description?     string
   |     +--rw static-routes
   |        +--rw v4ur:ipv4
   |        |     ...
   |        +--rw v6ur:ipv6
   |              ...
   +--rw ribs
      +--rw rib* [name]
         +--rw name              string
         +--rw address-family?   identityref
         +--ro default-rib?      boolean {multiple-ribs}?
         +--ro routes
         |  +--ro route*
         |        ...
         +---x active-route
         |  +---w input
         |  |  +---w v4ur:destination-address?   inet:ipv4-address
         |  |  +---w v6ur:destination-address?   inet:ipv6-address
         |  +--ro output
         |        ...
         +--rw description?      string
```

Figure 1: Data Hierarchy

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

4.1. System-Controlled and User-Controlled List Entries

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

In such a list, the server creates the required item as a so-called system-controlled entry in state data in the operational datastore [I-D.ietf-netmod-revised-datastores], i.e., inside read-only lists in the "routing" container.

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

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

Corresponding entries in both versions of the list (in operational datastore and intended datastore [I-D.ietf-netmod-revised-datastores] have the same value of the list key.

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

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

5. Basic Building Blocks

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

5.1. Route

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

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

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

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

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

5.2. Routing Information Base (RIB)

Every implementation of the core routing data model manages one or more Routing Information Bases (RIBs). A RIB is a list of routes complemented with administrative data. Each RIB contains only routes of one address family. An address family is represented by an identity derived from the "rt:address-family" base identity.

In the core routing data model, RIBs are state data represented as entries of the list "/routing/ribs/rib" in the operational datastore [I-D.ietf-netmod-revised-datastores]. The contents of RIBs are controlled and manipulated by control-plane protocol operations that may result in route additions, removals, and modifications. This also includes manipulations via the "static" and/or "direct" pseudo-protocols; see Section 5.3.1.

For every supported address family, exactly one RIB MUST be marked as the so-called default RIB to which control-plane protocols place their routes by default.

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

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

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

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

5.3. Control-Plane Protocol

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

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

5.3.1. Routing Pseudo-Protocols

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

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

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

5.3.2. Defining New Control-Plane Protocols

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

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

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

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

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

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

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

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

The above steps are implemented by the example YANG module for the Routing Information Protocol (RIP) in Appendix C.

5.4. Parameters of IPv6 Router Advertisements

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

- send-advertisements
- max-rtr-adv-interval
- min-rtr-adv-interval
- managed-flag
- other-config-flag
- link-mtu
- reachable-time
- retrans-timer
- cur-hop-limit

o default-lifetime

o prefix-list: a list of prefixes to be advertised.

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

* valid-lifetime

* on-link-flag

* preferred-lifetime

* autonomous-flag

NOTES:

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

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

6. Interactions with Other YANG Modules

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

6.1. Module "ietf-interfaces"

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

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

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

6.2. Module "ietf-ip"

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

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

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

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

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

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

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

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

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

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

7. Routing Management YANG Module

```
<CODE BEGINS> file "ietf-routing@2017-10-30.yang"
module ietf-routing {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-routing";
  prefix "rt";

  import ietf-yang-types {
```

```
       prefix "yang";
     }

     import ietf-interfaces {
       prefix "if";
     }

     organization
       "IETF NETMOD - Networking Modeling Working Group";
     contact
       "WG Web:   <http://tools.ietf.org/wg/netmod/>
        WG List:  <mailto:rtgwg@ietf.org>

        Editor:   Ladislav Lhotka
                  <mailto:lhotka@nic.cz>
                  Acee Lindem
                  <mailto:acee@cisco.com>
                  Yingzhen Qu
                  <mailto:yingzhen.qu@huawei.com>";

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

        Copyright (c) 2017 IETF Trust and the persons
        identified as authors of the code.  All rights reserved.

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

        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.";
     reference "RFC XXXX";

     revision 2017-10-30 {
       description
         "Network Managment Datastore Architecture (NDMA) Revision";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management
          (NDMA Version)";
     }

     revision 2016-11-04 {
          description
```

```
            "Initial revision.";
          reference
            "RFC 8022: A YANG Data Model for Routing Management";
     }

     /* Features */
     feature multiple-ribs {
       description
         "This feature indicates that the server supports user-defined
          RIBs.

          Servers that do not advertise this feature SHOULD provide
          exactly one system-controlled RIB per supported address family
          and make it also the default RIB.  This RIB then appears as an
          entry of the list /routing/ribs/rib.";
     }

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

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

     /* Identities */

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

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

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

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

   identity routing-protocol {
     base control-plane-protocol;
     description
       "Identity from which Layer 3 routing protocol identities are
        derived.";
   }

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

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

   /* Type Definitions */

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

   /* Groupings */

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

```
     }

     grouping router-id {
       description
         "This grouping provides router ID.";
       leaf router-id {
         type yang:dotted-quad;
         description
           "A 32-bit number in the form of a dotted quad that is used by
            some routing protocols identifying a router.";
         reference
           "RFC 2328: OSPF Version 2.";
       }
     }

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

     grouping next-hop-content {
```

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

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

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

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

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

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

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

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

                 Modules for address families MUST augment this list
                 with a leaf containing a next-hop address of that
                 address family.";
```

```
              leaf outgoing-interface {
                type if:interface-ref;
                description
                  "Name of the outgoing interface.";
              }
            }
          }
        }
      }
    }

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

    /* Configuration Data */

    container routing {
      description
        "Configuration parameters for the routing subsystem.";
      uses router-id {
        if-feature "router-id";
        description
          "Configuration of the global router ID.  Routing protocols
```

```
            that use router ID can use this parameter or override it
            with another value.";
       }
       container interfaces {
         config "false";
         description
           "Network-layer interfaces used for routing.";
         leaf-list interface {
           type if:interface-ref;
           description
             "Each entry is a reference to the name of a configured
              network-layer interface.";
         }
       }
       container control-plane-protocols {
         description
           "Configuration of control-plane protocol instances.";
         list control-plane-protocol {
           key "type name";
           description
             "Each entry contains configuration of a control-plane
              protocol instance.";
           leaf type {
             type identityref {
               base control-plane-protocol;
             }
             description
               "Type of the control-plane protocol - an identity derived
                from the 'control-plane-protocol' base identity.";
           }
           leaf name {
             type string;
             description
               "An arbitrary name of the control-plane protocol
                instance.";
           }
           leaf description {
             type string;
             description
               "Textual description of the control-plane protocol
                instance.";
           }
           container static-routes {
             when "derived-from-or-self(../type, 'rt:static')" {
               description
                 "This container is only valid for the 'static' routing
                  protocol.";
             }
```

```
            description
              "Configuration of the 'static' pseudo-protocol.

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

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

               For system-controlled entries, the value of this leaf
               must be the same as the name of the corresponding entry
               in state data.

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

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

          leaf default-rib {
            if-feature "multiple-ribs";
```

```
            type boolean;
            default "true";
            config "false";
            description
              "This flag has the value of 'true' if and only if the RIB
               is the default RIB for the given address family.

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

               Address-family-specific modules MUST augment input
               parameters with a leaf named 'destination-address'.";
            output {
              container route {
                description
                  "The active RIB route for the specified destination.

                   If no route exists in the RIB for the destination
```

```
                    address, no output is returned.

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

     /* Obsolete State Data */

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

            It may be either configured or assigned algorithmically by
            the implementation.";
       }
       container interfaces {
         status obsolete;
         description
           "Network-layer interfaces used for routing.";
         leaf-list interface {
           type if:interface-state-ref;
           status obsolete;
           description
             "Each entry is a reference to the name of a configured
              network-layer interface.";
         }
```

```
       }
       container control-plane-protocols {
         status obsolete;
         description
           "Container for the list of routing protocol instances.";
         list control-plane-protocol {
           key "type name";
           status obsolete;
           description
             "State data of a control-plane protocol instance.

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

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

```
               An implementation SHOULD provide one system-controlled
               default RIB for each supported address family.";
            leaf name {
              type string;
              status obsolete;
              description
                "The name of the RIB.";
            }
            uses address-family {
              status obsolete;
              description
                "The address family of the RIB.";
            }
            leaf default-rib {
              if-feature "multiple-ribs";
              type boolean;
              default "true";
              status obsolete;
              description
                "This flag has the value of 'true' if and only if the
                 RIB is the default RIB for the given address family.

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

```
                uses next-hop-state-content {
                  status obsolete;
                  description
                    "Route’s next-hop attribute operational state.";
                }
              }
              uses route-metadata {
                status obsolete;
                description
                  "Route metadata.";
              }
            }
          }
          action active-route {
            status obsolete;
            description
              "Return the active RIB route that is used for the
               destination address.

               Address-family-specific modules MUST augment input
               parameters with a leaf named ’destination-address’.";
            output {
              container route {
                status obsolete;
                description
                  "The active RIB route for the specified
                   destination.

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

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

```
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

8. IPv4 Unicast Routing Management YANG Module

```
  <CODE BEGINS> file "ietf-ipv4-unicast-routing@2017-10-14.yang"
  module ietf-ipv4-unicast-routing {
    yang-version "1.1";
    namespace
       "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing";
    prefix "v4ur";

    import ietf-routing {
      prefix "rt";
    }

    import ietf-inet-types {
      prefix "inet";
    }
    organization
      "IETF NETMOD - Networking Modeling Working Group";
    contact
      "WG Web:   <http://tools.ietf.org/wg/netmod/>
       WG List:  <mailto:rtgwg@ietf.org>

       Editor:   Ladislav Lhotka
                 <mailto:lhotka@nic.cz>
                 Acee Lindem
                 <mailto:acee@cisco.com>
                 Yingzhen Qu
                 <mailto:yingzhen.qu@huawei.com>";

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

       Copyright (c) 2017 IETF Trust and the persons
       identified as authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject
       to the license terms contained in, the Simplified BSD License
```

```
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.";
     reference "RFC XXXX";

     revision 2017-10-14 {
       description
         "Network Managment Datastore Architecture (NDMA) Revision";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management
          (NDMA Version)";
     }

     revision 2016-11-04 {
          description
            "Initial revision.";
          reference
            "RFC 8022: A YANG Data Model for Routing Management";
     }

     /* Identities */

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

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

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

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

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

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

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

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

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

```
         description
           "IPv4 address of the next hop.";
       }
     }

     augment "/rt:routing/rt:control-plane-protocols/"
           + "rt:control-plane-protocol/rt:static-routes" {
       description
         "This augment defines the configuration of the 'static'
          pseudo-protocol with data specific to IPv4 unicast.";
       container ipv4 {
         description
           "Configuration of a 'static' pseudo-protocol instance
            consists of a list of routes.";
         list route {
           key "destination-prefix";
           description
             "A list of static routes.";
           leaf destination-prefix {
             type inet:ipv4-prefix;
             mandatory "true";
             description
               "IPv4 destination prefix.";
           }
           leaf description {
             type string;
             description
               "Textual description of the route.";
           }
           container next-hop {
             description
               "Configuration of next-hop.";
             uses rt:next-hop-content {
               augment "next-hop-options/simple-next-hop" {
                 description
                   "Augment 'simple-next-hop' case in IPv4 static
                    routes.";
                 leaf next-hop-address {
                   type inet:ipv4-address;
                   description
                     "IPv4 address of the next hop.";
                 }
               }
               augment "next-hop-options/next-hop-list/next-hop-list/"
                     + "next-hop" {
                 description
                   "Augment 'next-hop-list' case in IPv4 static
                    routes.";
```

```
                 leaf next-hop-address {
                   type inet:ipv4-address;
                   description
                     "IPv4 address of the next hop.";
                 }
               }
             }
           }
         }
       }
     }

     /* Obsolete State Data */

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

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

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

9. IPv6 Unicast Routing Management YANG Module

```
<CODE BEGINS> file "ietf-ipv6-unicast-routing@2017-10-14.yang"
module ietf-ipv6-unicast-routing {
  yang-version "1.1";
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing";
  prefix "v6ur";

  import ietf-routing {
    prefix "rt";
  }

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

  include ietf-ipv6-router-advertisements {
    revision-date 2017-10-14;
  }

  organization
    "IETF NETMOD - Networking Modeling Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:rtgwg@ietf.org>

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>
               Acee Lindem
               <mailto:acee@cisco.com>
               Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>";

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

     Copyright (c) 2017 IETF Trust and the persons
     identified as authors of the code.  All rights reserved.

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

```
        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.";
     reference "RFC XXXX";

     revision 2017-10-14 {
       description
         "Network Managment Datastore Architecture (NDMA) revision";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management
          (NDMA Version)";
     }

     /* Identities */

     revision 2016-11-04 {
          description
            "Initial revision.";
          reference
            "RFC 8022: A YANG Data Model for Routing Management";
     }

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

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

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

```
       }
       description
         "Augment 'simple-next-hop' case in IPv6 unicast routes.";
       leaf next-hop-address {
         type inet:ipv6-address;
         description
           "IPv6 address of the next hop.";
       }
     }

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

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

     augment "/rt:routing/rt:ribs/rt:rib/rt:active-route/"
           + "rt:output/rt:route" {
       when "derived-from-or-self(../../rt:address-family, "
```

```
          + "'v6ur:ipv6-unicast')" {
         description
           "This augment is valid only for IPv6 unicast.";
       }
       description
         "This augment adds the destination prefix to the reply of the
          'active-route' action.";
       leaf destination-prefix {
         type inet:ipv6-prefix;
         description
           "IPv6 destination prefix.";
       }
     }

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

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

```
     }

     /* Configuration data */

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

```
                  type inet:ipv6-address;
                  description
                    "IPv6 address of the next hop.";
                }
              }
            }
          }
        }
      }
    }

    /* Obsolete State Data */

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

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

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

9.1. IPv6 Router Advertisements Submodule

```
<CODE BEGINS> file "ietf-ipv6-router-advertisements@2017-10-14.yang"
submodule ietf-ipv6-router-advertisements {
  yang-version "1.1";

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

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

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-ip {
    prefix "ip";
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

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

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

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

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

     Editor:   Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>";

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

     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).

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
        'OPTIONAL' in the module text are to be interpreted as
        described in RFC 2119.

        This version of this YANG module is part of RFC 8022;
        see the RFC itself for full legal notices.";

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

     revision 2017-10-14 {
       description
         "Network Managment Datastore Architecture (NDMA) Revision";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management
          (NDMA Version)";
     }

     revision 2016-11-04 {
          description
            "Initial revision.";
          reference
            "RFC 8022: A YANG Data Model for Routing Management";
     }

     augment "/if:interfaces/if:interface/ip:ipv6" {
       description
         "Augment interface configuration with parameters of IPv6
          router advertisements.";
       container ipv6-router-advertisements {
         description
           "Configuration of IPv6 Router Advertisements.";
         leaf send-advertisements {
           type boolean;
           default "false";
           description
             "A flag indicating whether or not the router sends
              periodic Router Advertisements and responds to
```

```
              Router Solicitations.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvSendAdvertisements.";
        }
        leaf max-rtr-adv-interval {
          type uint16 {
            range "4..1800";
          }
          units "seconds";
          default "600";
          description
            "The maximum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             MaxRtrAdvInterval.";
        }
        leaf min-rtr-adv-interval {
          type uint16 {
            range "3..1350";
          }
          units "seconds";
          must ". <= 0.75 * ../max-rtr-adv-interval" {
            description
              "The value MUST NOT be greater than 75% of
               'max-rtr-adv-interval'.";
          }
          description
            "The minimum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.

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

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

             - otherwise, it is 0.75 * max-rtr-adv-interval.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             MinRtrAdvInterval.";
        }
        leaf managed-flag {
          type boolean;
          default "false";
          description
            "The value to be placed in the 'Managed address
```

```
               configuration' flag field in the Router
               Advertisement.";
            reference
              "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
               AdvManagedFlag.";
          }
          leaf other-config-flag {
            type boolean;
            default "false";
            description
              "The value to be placed in the 'Other configuration'
               flag field in the Router Advertisement.";
            reference
              "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
               AdvOtherConfigFlag.";
          }
          leaf link-mtu {
            type uint32;
            default "0";
            description
              "The value to be placed in MTU options sent by the
               router. A value of zero indicates that no MTU options
               are sent.";
            reference
              "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
               AdvLinkMTU.";
          }
          leaf reachable-time {
            type uint32 {
              range "0..3600000";
            }
            units "milliseconds";
            default "0";
            description
              "The value to be placed in the Reachable Time field in
               the Router Advertisement messages sent by the router.
               A value of zero means unspecified (by this router).";
            reference
              "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
               AdvReachableTime.";
          }
          leaf retrans-timer {
            type uint32;
            units "milliseconds";
            default "0";
            description
              "The value to be placed in the Retrans Timer field in
               the Router Advertisement messages sent by the router.
```

```
              A value of zero means unspecified (by this router).";
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvRetransTimer.";
         }
         leaf cur-hop-limit {
           type uint8;
           description
             "The value to be placed in the Cur Hop Limit field in
              the Router Advertisement messages sent by the router.
              A value of zero means unspecified (by this router).

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

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

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

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

                 The link-local prefix SHOULD NOT be included in the
                 list of advertised prefixes.";
              reference
                "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
                 AdvPrefixList.";
              list prefix {
                key "prefix-spec";
                description
                  "Configuration of an advertised prefix entry.";
                leaf prefix-spec {
                  type inet:ipv6-prefix;
                  description
                    "IPv6 address prefix.";
                }
                choice control-adv-prefixes {
                  default "advertise";
                  description
                    "Either the prefix is explicitly removed from the
                     set of advertised prefixes, or the parameters with
                     which it is advertised are specified (default
                     case).";
                  leaf no-advertise {
                    type empty;
                    description
                      "The prefix will not be advertised.

                       This can be used for removing the prefix from
                       the default set of advertised prefixes.";
                  }
                  case advertise {
                    leaf valid-lifetime {
                      type uint32;
                      units "seconds";
                      default "2592000";
                      description
                        "The value to be placed in the Valid Lifetime
                         in the Prefix Information option.  The
                         designated value of all 1's (0xffffffff)
                          represents infinity.";
                      reference
                        "RFC 4861: Neighbor Discovery for IP version 6
                         (IPv6) - AdvValidLifetime.";
                    }
                    leaf on-link-flag {
```

```
                  type boolean;
                  default "true";
                  description
                    "The value to be placed in the on-link flag
                     ('L-bit') field in the Prefix Information
                     option.";
                  reference
                    "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvOnLinkFlag.";
                }
                leaf preferred-lifetime {
                  type uint32;
                  units "seconds";
                  must ". <= ../valid-lifetime" {
                    description
                      "This value MUST NOT be greater than
                       valid-lifetime.";
                  }
                  default "604800";
                  description
                    "The value to be placed in the Preferred
                     Lifetime in the Prefix Information option.
                     The designated value of all 1's (0xffffffff)
                     represents infinity.";
                  reference
                    "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvPreferredLifetime.";
                }
                leaf autonomous-flag {
                  type boolean;
                  default "true";
                  description
                    "The value to be placed in the Autonomous Flag
                     field in the Prefix Information option.";
                  reference
                    "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvAutonomousFlag.";
                }
              }
            }
          }
        }
      }
    }

    /* Obsolete State Data */

    augment "/if:interfaces-state/if:interface/ip:ipv6" {
```

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

```
               field in the Router Advertisement.";
          }
          leaf link-mtu {
            type uint32;
            status obsolete;
            description
              "The value that is placed in MTU options sent by the
               router.  A value of zero indicates that no MTU options are
               sent.";
          }
          leaf reachable-time {
            type uint32 {
              range "0..3600000";
            }
            units "milliseconds";
            status obsolete;
            description
              "The value that is placed in the Reachable Time field in
               the Router Advertisement messages sent by the router.  A
               value of zero means unspecified (by this router).";
          }
          leaf retrans-timer {
            type uint32;
            units "milliseconds";
            status obsolete;
            description
              "The value that is placed in the Retrans Timer field in the
               Router Advertisement messages sent by the router.  A value
               of zero means unspecified (by this router).";
          }
          leaf cur-hop-limit {
            type uint8;
            status obsolete;
            description
              "The value that is placed in the Cur Hop Limit field in the
               Router Advertisement messages sent by the router.  A value
               of zero means unspecified (by this router).";
          }
          leaf default-lifetime {
            type uint16 {
              range "0..9000";
            }
            units "seconds";
            status obsolete;
            description
              "The value that is placed in the Router Lifetime field of
               Router Advertisements sent from the interface, in seconds.
               A value of zero indicates that the router is not to be
```

```
                used as a default router.";
          }
          container prefix-list {
            status obsolete;
            description
              "A list of prefixes that are placed in Prefix Information
               options in Router Advertisement messages sent from the
               interface.

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

                   An implementation SHOULD keep this value constant in
                   consecutive advertisements except when it is
                   explicitly changed in configuration.";
              }
              leaf on-link-flag {
                type boolean;
                status obsolete;
                description
                  "The value that is placed in the on-link flag ('L-bit')
                   field in the Prefix Information option.";
              }
              leaf preferred-lifetime {
                type uint32;
                units "seconds";
                status obsolete;
                description
```

```
                 "The value that is placed in the Preferred Lifetime in
                  the Prefix Information option, in seconds.  The
                  designated value of all 1's (0xffffffff) represents
                  infinity.

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

10. IANA Considerations

[RFC8022] registered the following namespace URIs in the "IETF XML Registry" [RFC3688]:

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

URI: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

[RFC8022] registered the following YANG modules in the "YANG Module Names" registry [RFC6020]:

```
Name:          ietf-routing
Namespace:     urn:ietf:params:xml:ns:yang:ietf-routing
Prefix:        rt
Reference:     RFC 8022
```

```
Name:         ietf-ipv4-unicast-routing
Namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
Prefix:       v4ur
Reference:    RFC 8022

Name:         ietf-ipv6-unicast-routing
Namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
Prefix:       v6ur
Reference:    RFC 8022
```

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

```
Name:         ietf-ipv6-router-advertisements
Module:       ietf-ipv6-unicast-routing
Reference:    RFC 8022
```

11. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242].

There are a number of data nodes defined in this YANG module which 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:

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

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

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

/routing/control-plane-protocols/control-plane-protocol: This list specifies the control-plane protocols configured on a device. Refer to the control plane models for a list of sensitive information.

/routing/ribs/rib: This list specifies the RIB and their contents for the device. Access to this information may disclose the network topology and or other information.

12. References

12.1. Normative References

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

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

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

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

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

[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>.

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

[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>.

[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>.

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <https://www.rfc-editor.org/info/rfc8022>.

12.2. Informative References

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

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

[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>.

[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-05 (work in progress), October 2017.

Appendix A. The Complete Data Trees

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

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

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

```
           |  |  +---w v6ur:destination-address?   inet:ipv6-address
           |  +--ro output
           |     +--ro route
           |        +--ro next-hop
           |        |  +--ro (next-hop-options)
           |        |     +--:(simple-next-hop)
           |        |     |  +--ro outgoing-interface?
           |        |     |  |       if:interface-ref
           |        |     |  +--ro v4ur:next-hop-address?
           |        |     |  |          inet:ipv4-address
           |        |     |  +--ro v6ur:next-hop-address?
           |        |     |             inet:ipv6-address
           |        |     +--:(special-next-hop)
           |        |     |  +--ro special-next-hop?      enumeration
           |        |     +--:(next-hop-list)
           |        |        +--ro next-hop-list
           |        |           +--ro next-hop*
           |        |              +--ro outgoing-interface?
           |        |              |       if:interface-ref
           |        |              +--ro v4ur:next-hop-address?
           |        |              |       inet:ipv4-address
           |        |              +--ro v6ur:next-hop-address?
           |        |                      inet:ipv6-address
           |        +--ro source-protocol          identityref
           |        +--ro active?                  empty
           |        +--ro last-updated?            yang:date-and-time
           |        +--ro v4ur:destination-prefix? inet:ipv4-prefix
           |        +--ro v6ur:destination-prefix? inet:ipv6-prefix
           +--rw description?      string
   module: ietf-ipv6-unicast-routing
     augment /if:interfaces/if:interface/ip:ipv6:
   +--rw ipv6-router-advertisements
      +--rw send-advertisements?    boolean
      +--rw max-rtr-adv-interval?   uint16
      +--rw min-rtr-adv-interval?   uint16
      +--rw managed-flag?           boolean
      +--rw other-config-flag?      boolean
      +--rw link-mtu?               uint32
      +--rw reachable-time?         uint32
      +--rw retrans-timer?          uint32
      +--rw cur-hop-limit?          uint8
      +--rw default-lifetime?       uint16
      +--rw prefix-list
         +--rw prefix* [prefix-spec]
            +--rw prefix-spec           inet:ipv6-prefix
            +--rw (control-adv-prefixes)?
               +--:(no-advertise)
               |  +--rw no-advertise?         empty
```

```
               +--:(advertise)
                  +--rw valid-lifetime?       uint32
                  +--rw on-link-flag?         boolean
                  +--rw preferred-lifetime?   uint32
                  +--rw autonomous-flag?      boolean
```

Appendix B. Minimum Implementation

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

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

In addition to the mandatory instance of the "direct" pseudo-protocol, a minimum implementation should support configuring instance(s) of the "static" pseudo-protocol.

For hosts that are never intended to act as routers, the ability to turn on sending IPv6 router advertisements (Section 5.4) should be removed.

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" control-plane protocol instances.

Appendix C. Example: Adding a New Control-Plane Protocol

This appendix demonstrates how the core routing data model can be extended to support a new control-plane protocol. The YANG module "example-rip" shown below is intended as an illustration rather than a real definition of a data model for the Routing Information Protocol (RIP). For the sake of brevity, this module does not obey all the guidelines specified in [RFC6087]. See also Section 5.3.2.

```
module example-rip {

  yang-version "1.1";

  namespace "http://example.com/rip";

  prefix "rip";
```

```
     import ietf-interfaces {
       prefix "if";
     }

     import ietf-routing {
       prefix "rt";
     }

     identity rip {
       base rt:routing-protocol;
       description
         "Identity for the Routing Information Protocol (RIP).";
     }

     typedef rip-metric {
       type uint8 {
         range "0..16";
       }
     }

     grouping route-content {
       description
         "This grouping defines RIP-specific route attributes.";
       leaf metric {
         type rip-metric;
       }
       leaf tag {
         type uint16;
         default "0";
         description
           "This leaf may be used to carry additional info, e.g.,
            autonomous system (AS) number.";
       }
     }

     augment "/rt:routing/rt:ribs/rt:rib/rt:routes/rt:route" {
       when "derived-from-or-self(rt:source-protocol, 'rip:rip')" {
         description
           "This augment is only valid for a route whose source
            protocol is RIP.";
       }
       description
         "RIP-specific route attributes.";
       uses route-content;
     }

     augment "/rt:routing/rt:ribs/rt:rib/rt:active-route/"
           + "rt:output/rt:route" {
```

```
       description
         "RIP-specific route attributes in the output of 'active-route'
          RPC.";
       uses route-content;
     }

     augment "/rt:routing/rt:control-plane-protocols/"
           + "rt:control-plane-protocol" {
       when "derived-from-or-self(rt:type,'rip:rip')" {
         description
           "This augment is only valid for a routing protocol instance
            of type 'rip'.";
       }
       container rip {
         presence "RIP configuration";
         description
           "RIP instance configuration.";
         container interfaces {
           description
             "Per-interface RIP configuration.";
           list interface {
             key "name";
             description
               "RIP is enabled on interfaces that have an entry in this
                list, unless 'enabled' is set to 'false' for that
                entry.";
             leaf name {
               type if:interface-ref;
             }
             leaf enabled {
               type boolean;
               default "true";
             }
             leaf metric {
               type rip-metric;
               default "1";
             }
           }
         }
         leaf update-interval {
           type uint8 {
             range "10..60";
           }
           units "seconds";
           default "30";
           description
             "Time interval between periodic updates.";
         }
```

```
      }
    }
  }
```

Appendix D. Data Tree Example

This section contains an example of an instance data tree in the JSON encoding [RFC7951], containing both configuration and state data. The data conforms to a data model that is defined by the following YANG library specification [RFC7895]:


```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "c2e1f54169aa7f36e1a6e8d0865d441d3600f9c4",
    "module": [
      {
        "name": "ietf-routing",
        "revision": "2017-09-13",
        "feature": [
          "multiple-ribs",
          "router-id"
        ],
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv4-unicast-routing",
        "revision": "2017-09-13",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-ipv6-unicast-routing",
        "revision": "2017-09-13",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing-3",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-interfaces",
        "revision": "2014-05-08",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-inet-types",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
```

```
          "revision": "2013-07-15",
          "conformance-type": "import"
        },
        {
          "name": "ietf-yang-types",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
          "revision": "2013-07-15",
          "conformance-type": "import"
        },
        {
          "name": "iana-if-type",
          "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
          "revision": "",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        }
      ]
    }
  }
```


A simple network setup as shown in Figure 2 is assumed: router "A" uses static default routes with the "ISP" router as the next hop. IPv6 router advertisements are configured only on the "eth1" interface and disabled on the upstream "eth0" interface.

```
                  +-----------------+
                  |                 |
                  |    Router ISP   |
                  |                 |
                  +--------+--------+
                           |2001:db8:0:1::2
                           |192.0.2.2
                           |
                           |
                           |2001:db8:0:1::1
                       eth0|192.0.2.1
                  +--------+--------+
                  |                 |
                  |     Router A    |
                  |                 |
                  +--------+--------+
                       eth1|198.51.100.1
                           |2001:db8:0:2::1
                           |
```

Figure 2: Example of Network Configuration

The instance data tree could then be as follows:


```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "description": "Uplink to ISP.",
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.1",
              "prefix-length": 24
            }
          ],
          "forwarding": true
        },
        "ietf-ip:ipv6": {
          "address": [
            {
              "ip": "2001:0db8:0:1::1",
              "prefix-length": 64
            }
          ],
          "forwarding": true,
```

```
            "autoconf": {
              "create-global-addresses": false
            }
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "description": "Interface to the internal network.",
          "ietf-ip:ipv4": {
            "address": [
              {
                "ip": "198.51.100.1",
                "prefix-length": 24
              }
            ],
            "forwarding": true
          },
          "ietf-ip:ipv6": {
            "address": [
              {
                "ip": "2001:0db8:0:2::1",
                "prefix-length": 64
              }
            ],
            "forwarding": true,
            "autoconf": {
              "create-global-addresses": false
            },
            "ietf-ipv6-unicast-routing:
                ipv6-router-advertisements": {
              "send-advertisements": true
            }
          }
        }
      ]
    },
    "ietf-interfaces:interfaces-state": {
      "interface": [
        {
          "name": "eth0",
          "type": "iana-if-type:ethernetCsmacd",
          "phys-address": "00:0C:42:E5:B1:E9",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2015-10-24T17:11:27+02:00"
          },
          "ietf-ip:ipv4": {
```

```
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "192.0.2.1",
                "prefix-length": 24
              }
            ]
          },
          "ietf-ip:ipv6": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:1::1",
                "prefix-length": 64
              }
            ],
            "ietf-ipv6-unicast-routing:
               ipv6-router-advertisements": {
              "send-advertisements": false
            }
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "phys-address": "00:0C:42:E5:B1:EA",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2015-10-24T17:11:29+02:00"
          },
          "ietf-ip:ipv4": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "198.51.100.1",
                "prefix-length": 24
              }
            ]
          },
          "ietf-ip:ipv6": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:2::1",
```

```
                "prefix-length": 64
              }
            ],
            "ietf-ipv6-unicast-routing:
               ipv6-router-advertisements": {
              "send-advertisements": true,
              "prefix-list": {
                "prefix": [
                  {
                    "prefix-spec": "2001:db8:0:2::/64"
                  }
                ]
              }
            }
          }
        }
      ]
    },
    "ietf-routing:routing": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:static",
            "name": "st0",
            "description":
              "Static routing is used for the internal network.",
            "static-routes": {
              "ietf-ipv4-unicast-routing:ipv4": {
                "route": [
                  {
                    "destination-prefix": "0.0.0.0/0",
                    "next-hop": {
                      "next-hop-address": "192.0.2.2"
                    }
                  }
                ]
              },
              "ietf-ipv6-unicast-routing:ipv6": {
                "route": [
                  {
                    "destination-prefix": "::/0",
                    "next-hop": {
                      "next-hop-address": "2001:db8:0:1::2"
                    }
                  }
                ]
              }
```

```
            }
          }
        ]
      }
          "ribs": {
        "rib": [
          {
            "name": "ipv4-master",
            "address-family":
              "ietf-ipv4-unicast-routing:ipv4-unicast",
            "default-rib": true,
            "routes": {
              "route": [
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "192.0.2.1/24",
                  "next-hop": {
                    "outgoing-interface": "eth0"
                  },
                  "route-preference": 0,
                  "source-protocol": "ietf-routing:direct",
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "198.51.100.0/24",
                  "next-hop": {
                    "outgoing-interface": "eth1"
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv4-unicast-routing:destination-prefix":
                    "0.0.0.0/0",
                  "source-protocol": "ietf-routing:static",
                  "route-preference": 5,
                  "next-hop": {
                    "ietf-ipv4-unicast-routing:next-hop-address":
                      "192.0.2.2"
                  },
                  "last-updated": "2015-10-24T18:02:45+02:00"
                }
              ]
            }
          },
          {
```

```
            "name": "ipv6-master",
            "address-family":
              "ietf-ipv6-unicast-routing:ipv6-unicast",
            "default-rib": true,
            "routes": {
              "route": [
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
                    "2001:db8:0:1::/64",
                  "next-hop": {
                    "outgoing-interface": "eth0"
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
                    "2001:db8:0:2::/64",
                  "next-hop": {
                    "outgoing-interface": "eth1"
                  },
                  "source-protocol": "ietf-routing:direct",
                  "route-preference": 0,
                  "last-updated": "2015-10-24T17:11:27+02:00"
                },
                {
                  "ietf-ipv6-unicast-routing:destination-prefix":
                    "::/0",
                  "next-hop": {
                    "ietf-ipv6-unicast-routing:next-hop-address":
                      "2001:db8:0:1::2"
                  },
                  "source-protocol": "ietf-routing:static",
                  "route-preference": 5,
                  "last-updated": "2015-10-24T18:02:45+02:00"
                }
              ]
            }
          }
        ]
      }
    },
  }
```

Acknowledgments

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

Authors' Addresses

Ladislav Lhotka
CZ.NIC

EMail: lhotka@nic.cz

Acee Lindem
Cisco Systems

EMail: acee@cisco.com

Yingzhen Qu
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara CA 95050
USA

EMail: yingzhen.qu@huawei.com

NETMOD Working Group
Internet-Draft
Obsoletes: 8022 (if approved)
Intended status: Standards Track
Expires: July 30, 2018

L. Lhotka
CZ.NIC
A. Lindem
Cisco Systems
Y. Qu
Huawei

January 26, 2018

A YANG Data Model for Routing Management (NMDA Version)
draft-ietf-netmod-rfc8022bis-11

Abstract

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

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA). This document obsoletes RFC 8022.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document contains a specification of the following YANG modules:

- o The "ietf-routing" module provides generic components of a routing data model.

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

- o The "ietf-ipv6-unicast-routing" module augments the "ietf-routing" module with additional data specific to IPv6 unicast. Its submodule "ietf-ipv6-router-advertisements" also augments the "ietf-interfaces" [I-D.ietf-netmod-rfc7223bis] and "ietf-ip" [I-D.ietf-netmod-rfc7277bis] modules with IPv6 router configuration variables required by [RFC4861].

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

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [I-D.ietf-netmod-revised-datastores]. This document obsoletes RFC 8022 [RFC8022].

2. Terminology and Notation

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

The following terms are defined in [I-D.ietf-netmod-revised-datastores]:

- o client

- o server

- o configuration
- o system state
- o operational state
- o intended configuration

The following terms are defined in [RFC7950]:

- o action
- o augment
- o container
- o container with presence
- o data model
- o data node
- o feature
- o leaf
- o list
- o mandatory node
- o module
- o schema tree
- o RPC (Remote Procedure Call) operation

2.1. Glossary of New Terms

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

direct route: a route to a directly connected network.

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

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

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

2.2. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [I-D.ietf-netmod-yang-tree-diagrams].

2.3. Prefixes in Data Node Names

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

Prefix	YANG module	Reference
if	ietf-interfaces	[I-D.ietf-netmod-rfc7223bis]
ip	ietf-ip	[I-D.ietf-netmod-rfc7277bis]
rt	ietf-routing	Section 7
v4ur	ietf-ipv4-unicast-routing	Section 8
v6ur	ietf-ipv6-unicast-routing	Section 9
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]

Table 1: Prefixes and Corresponding YANG Modules

3. Objectives

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

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

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

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

- Because device vendors will want to map the data models built on this generic framework to their proprietary data models and configuration interfaces, the framework should be flexible enough to facilitate that and accommodate data models with different logic.

4. The Design of the Core Routing Data Model

The core routing data model consists of three YANG modules and one submodule. The first module, "ietf-routing", defines the generic components of a routing system. The other two modules, "ietf-ipv4-unicast-routing" and "ietf-ipv6-unicast-routing", augment the "ietf-routing" module with additional data nodes that are needed for IPv4 and IPv6 unicast routing, respectively. The "ietf-ipv6-unicast-routing" module has a submodule, "ietf-ipv6-router-advertisements", that augments the "ietf-interfaces" [I-D.ietf-netmod-rfc7223bis] and "ietf-ip" [I-D.ietf-netmod-rfc7277bis] modules with configuration variables for IPv6 router advertisements as required by [RFC4861].

Figure 1 shows abridged views of the hierarchies. See Appendix A for the complete data trees.

```
   +--rw routing
      +--rw router-id?                 yang:dotted-quad
      +--ro interfaces
      |  +--ro interface*   if:interface-ref
      +--rw control-plane-protocols
      |  +--rw control-plane-protocol* [type name]
      |     +--rw type            identityref
      |     +--rw name            string
      |     +--rw description?    string
      |     +--rw static-routes
      |        +--rw v4ur:ipv4
      |        |     ...
      |        +--rw v6ur:ipv6
      |              ...
      +--rw ribs
         +--rw rib* [name]
            +--rw name              string
            +--rw address-family?   identityref
            +--ro default-rib?      boolean {multiple-ribs}?
            +--ro routes
            |  +--ro route*
            |        ...
            +---x active-route
            |  +---w input
            |  |  +---w v4ur:destination-address?   inet:ipv4-address
            |  |  +---w v6ur:destination-address?   inet:ipv6-address
            |  +--ro output
            |        ...
            +--rw description?      string
```

Figure 1: Data Hierarchy

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

4.1. System-Controlled and User-Controlled List Entries

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

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

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

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

Corresponding entries in both versions of the list (in the intended configuration and the operational state) [I-D.ietf-netmod-revised-datastores] have the same value of the list key.

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

Deleting a user-controlled entry from the intended configuration results in the removal of the corresponding entry in the operational state list. In contrast, if client deletes a system-controlled entry from the intended configuration, only the extra configuration specified in that entry is removed but the corresponding operational state entry is not removed.

5. Basic Building Blocks

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

5.1. Route

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

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

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

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

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

5.2. Routing Information Base (RIB)

Every implementation of the core routing data model manages one or more Routing Information Bases (RIBs). A RIB is a list of routes complemented with administrative data. Each RIB contains only routes of one address family. An address family is represented by an identity derived from the "rt:address-family" base identity.

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

For every supported address family, exactly one RIB MUST be marked as the so-called default RIB to which control-plane protocols place their routes by default.

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

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

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

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

5.3. Control-Plane Protocol

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

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

5.3.1. Routing Pseudo-Protocols

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

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

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

5.3.2. Defining New Control-Plane Protocols

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

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

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

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

  and possibly other places in the schema tree.

- Data nodes for the new protocol can be defined by augmenting the "control-plane-protocol" data node under "/routing".

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

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

The above steps are implemented by the example YANG module for the Routing Information Protocol (RIP) in Appendix C.

5.4. Parameters of IPv6 Router Advertisements

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

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

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

  * valid-lifetime
  * on-link-flag
  * preferred-lifetime

   * autonomous-flag

NOTES:

1. The "IsRouter" flag, which is also required by [RFC4861], is implemented in the "ietf-ip" module [I-D.ietf-netmod-rfc7277bis] (leaf "ip:forwarding").

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

6. Interactions with Other YANG Modules

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

6.1. Module "ietf-interfaces"

The following boolean switch is defined in the "ietf-interfaces" YANG module [I-D.ietf-netmod-rfc7223bis]:

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

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

6.2. Module "ietf-ip"

The following boolean switches are defined in the "ietf-ip" YANG module [I-D.ietf-netmod-rfc7277bis]:

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

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

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

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

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

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

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

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

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

7. Routing Management YANG Module

```
<CODE BEGINS> file "ietf-routing@2018-01-25.yang"
module ietf-routing {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-routing";
  prefix "rt";

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

  import ietf-interfaces {
    prefix "if";
    description
      "A Network Management Datastore Architecture (NMDA)
       compatible version of the ietf-interfaces module
       is required.";
  }

  organization
    "IETF NETMOD - Networking Modeling Working Group";
```

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

      Editor:   Ladislav Lhotka
                <mailto:lhotka@nic.cz>
                Acee Lindem
                <mailto:acee@cisco.com>
                Yingzhen Qu
                <mailto:yingzhen.qu@huawei.com>";

   description
     "This YANG module defines essential components for the management
      of a routing subsystem. The model fully conforms to 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 RFC XXXX; see
      the RFC itself for full legal notices.";
   reference "RFC XXXX";

   revision 2018-01-25 {
     description
       "Network Management Datastore Architecture (NMDA) Revision";
     reference
       "RFC XXXX: A YANG Data Model for Routing Management
        (NMDA Version)";
   }

   revision 2016-11-04 {
        description
          "Initial revision.";
        reference
          "RFC 8022: A YANG Data Model for Routing Management";
   }

   /* Features */
   feature multiple-ribs {
     description
```

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

          Servers that do not advertise this feature SHOULD provide
          exactly one system-controlled RIB per supported address family
          and make it also the default RIB.  This RIB then appears as an
          entry of the list /routing/ribs/rib.";
     }

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

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

     /* Identities */

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

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

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

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

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

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

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

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

     /* Type Definitions */

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

     /* Groupings */

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

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

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

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

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

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

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

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

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

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

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

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

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

```
     }

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

     /* Data nodes */

     container routing {
       description
         "Configuration parameters for the routing subsystem.";
       uses router-id {
         if-feature "router-id";
         description
           "Support for the global router ID.  Routing protocols
            that use router ID can use this parameter or override it
            with another value.";
       }
       container interfaces {
         config "false";
         description
           "Network-layer interfaces used for routing.";
         leaf-list interface {
           type if:interface-ref;
```

```
          description
            "Each entry is a reference to the name of a configured
             network-layer interface.";
        }
      }
      container control-plane-protocols {
        description
          "Support for control-plane protocol instances.";
        list control-plane-protocol {
          key "type name";
          description
            "Each entry contains a control-plane protocol instance.";
          leaf type {
            type identityref {
              base control-plane-protocol;
            }
            description
              "Type of the control-plane protocol - an identity derived
               from the 'control-plane-protocol' base identity.";
          }
          leaf name {
            type string;
            description
              "An arbitrary name of the control-plane protocol
               instance.";
          }
          leaf description {
            type string;
            description
              "Textual description of the control-plane protocol
               instance.";
          }
          container static-routes {
            when "derived-from-or-self(../type, 'rt:static')" {
              description
                "This container is only valid for the 'static' routing
                 protocol.";
            }
            description
              "Support for the 'static' pseudo-protocol.

               Address-family-specific modules augment this node with
               their lists of routes.";
          }
        }
      }
      container ribs {
        description
```

```
          "Support for RIBs.";
        list rib {
          key "name";
          description
            "Each entry contains configuration for a RIB identified by
             the 'name' key.

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

               For system-controlled entries, the value of this leaf
               must be the same as the name of the corresponding entry
               in operational state.

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

          leaf default-rib {
            if-feature "multiple-ribs";
            type boolean;
            default "true";
            config "false";
            description
              "This flag has the value of 'true' if and only if the RIB
               is the default RIB for the given address family.

               By default, control-plane protocols place their routes
               in the default RIBs.";
          }
          container routes {
            config "false";
            description
              "Current content of the RIB.";
            list route {
              description
                "A RIB route entry.  This data node MUST be augmented
                 with information specific for routes of each address
```

```
                family.";
             leaf route-preference {
               type route-preference;
               description
                 "This route attribute, also known as administrative
                  distance, allows for selecting the preferred route
                  among routes with the same destination prefix.  A
                  smaller value means a more preferred route.";
             }
             container next-hop {
               description
                 "Route’s next-hop attribute.";
               uses next-hop-state-content;
             }
             uses route-metadata;
           }
         }
         action active-route {
           description
             "Return the active RIB route that is used for the
              destination address.

              Address-family-specific modules MUST augment input
              parameters with a leaf named ’destination-address’.";
           output {
             container route {
               description
                 "The active RIB route for the specified destination.

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

                  Address-family-specific modules MUST augment this
                  container with appropriate route contents.";
               container next-hop {
                 description
                   "Route’s next-hop attribute.";
                 uses next-hop-state-content;
               }
               uses route-metadata;
             }
           }
         }
         leaf description {
           type string;
           description
             "Textual description of the RIB.";
         }
```

```
        }
      }
    }

    /*
     * The subsequent data nodes are obviated and obsoleted by the
     * "Network Management Architecture" as described in
     * draft-ietf-netmod-revised-datastores.
     */
    container routing-state {
      config false;
      status obsolete;
      description
        "State data of the routing subsystem.";
      uses router-id {
        status obsolete;
        description
          "Global router ID.

           It may be either configured or assigned algorithmically by
           the implementation.";
      }
      container interfaces {
        status obsolete;
        description
          "Network-layer interfaces used for routing.";
        leaf-list interface {
          type if:interface-state-ref;
          status obsolete;
          description
            "Each entry is a reference to the name of a configured
             network-layer interface.";
        }
      }
      container control-plane-protocols {
        status obsolete;
        description
          "Container for the list of routing protocol instances.";
        list control-plane-protocol {
          key "type name";
          status obsolete;
          description
            "State data of a control-plane protocol instance.

             An implementation MUST provide exactly one
             system-controlled instance of the 'direct'
             pseudo-protocol.  Instances of other control-plane
             protocols MAY be created by configuration.";
```

```
          leaf type {
            type identityref {
              base control-plane-protocol;
            }
            status obsolete;
            description
              "Type of the control-plane protocol.";
          }
          leaf name {
            type string;
            status obsolete;
            description
              "The name of the control-plane protocol instance.

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

             An implementation SHOULD provide one system-controlled
             default RIB for each supported address family.";
          leaf name {
            type string;
            status obsolete;
            description
              "The name of the RIB.";
          }
          uses address-family {
            status obsolete;
            description
              "The address family of the RIB.";
          }
          leaf default-rib {
            if-feature "multiple-ribs";
```

```
             type boolean;
             default "true";
             status obsolete;
             description
               "This flag has the value of 'true' if and only if the
                RIB is the default RIB for the given address family.

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

```
             description
               "Return the active RIB route that is used for the
                destination address.

                Address-family-specific modules MUST augment input
                parameters with a leaf named 'destination-address'.";
             output {
               container route {
                 status obsolete;
                 description
                   "The active RIB route for the specified
                    destination.

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

                    Address-family-specific modules MUST augment this
                    container with appropriate route contents.";
                 container next-hop {
                   status obsolete;
                   description
                     "Route's next-hop attribute.";
                   uses next-hop-state-content {
                     status obsolete;
                     description
                       "Active route state data.";
                   }
                 }
                 uses route-metadata {
                   status obsolete;
                   description
                   "Active route metadata.";
                 }
               }
             }
           }
         }
       }
     }
   }
   <CODE ENDS>
```

8. IPv4 Unicast Routing Management YANG Module

```
   <CODE BEGINS> file "ietf-ipv4-unicast-routing@2018-01-25.yang"
   module ietf-ipv4-unicast-routing {
     yang-version "1.1";
     namespace
```

```
        "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing";
      prefix "v4ur";

      import ietf-routing {
        prefix "rt";
        description
          "A Network Management Datastore Architecture (NMDA)
           compatible version of the ietf-routing module
           is required.";
      }

      import ietf-inet-types {
        prefix "inet";
      }
      organization
        "IETF NETMOD - Networking Modeling Working Group";
      contact
        "WG Web:   <http://tools.ietf.org/wg/netmod/>
         WG List:  <mailto:rtgwg@ietf.org>

         Editor:   Ladislav Lhotka
                   <mailto:lhotka@nic.cz>
                   Acee Lindem
                   <mailto:acee@cisco.com>
                   Yingzhen Qu
                   <mailto:yingzhen.qu@huawei.com>";

      description
        "This YANG module augments the 'ietf-routing' module with basic
         parameters for IPv4 unicast routing. The model fully conforms
         to 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 RFC XXXX; see
         the RFC itself for full legal notices.";
      reference "RFC XXXX";

      revision 2018-01-25 {
        description
```

```
         "Network Management Datastore Architecture (NMDA) Revision";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management
          (NMDA Version)";
     }

     revision 2016-11-04 {
          description
            "Initial revision.";
          reference
            "RFC 8022: A YANG Data Model for Routing Management";
     }

     /* Identities */

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

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

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

```
             "IPv4 address of the next hop.";
         }
       }

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

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

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

```
          'active-route' action.";
       leaf destination-prefix {
         type inet:ipv4-prefix;
         description
           "IPv4 destination prefix.";
       }
     }

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

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

     augment "/rt:routing/rt:control-plane-protocols/"
           + "rt:control-plane-protocol/rt:static-routes" {
       description
         "This augment defines the 'static' pseudo-protocol
```

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

```
      }
    }

    /*
     * The subsequent data nodes are obviated and obsoleted by the
     * "Network Management Architecture" as described in
     * draft-ietf-netmod-revised-datastores.
     */
    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
      when "derived-from-or-self(../../rt:address-family, "
           + "'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      status obsolete;
      description
        "This leaf augments an IPv4 unicast route.";
      leaf destination-prefix {
        type inet:ipv4-prefix;
        status obsolete;
        description
          "IPv4 destination prefix.";
      }
    }
    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
      when "derived-from-or-self(
              ../../../rt:address-family, 'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
      }
      status obsolete;
      description
        "Augment 'simple-next-hop' case in IPv4 unicast routes.";
      leaf next-hop-address {
        type inet:ipv4-address;
        status obsolete;
        description
          "IPv4 address of the next hop.";
      }
    }
    augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
          + "rt:next-hop/rt:next-hop-options/rt:next-hop-list/"
          + "rt:next-hop-list/rt:next-hop" {
      when "derived-from-or-self(../../../../../rt:address-family,
              'v4ur:ipv4-unicast')" {
        description
          "This augment is valid only for IPv4 unicast.";
```

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

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

9. IPv6 Unicast Routing Management YANG Module

```
   <CODE BEGINS> file "ietf-ipv6-unicast-routing@2018-01-25.yang"
   module ietf-ipv6-unicast-routing {
     yang-version "1.1";
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing";
```

```
   prefix "v6ur";

   import ietf-routing {
     prefix "rt";
     description
       "A Network Management Datastore Architecture (NMDA)
        compatible version of the ietf-routing module
        is required.";
   }

   import ietf-inet-types {
     prefix "inet";
     description
       "A Network Management Datastore Architecture (NMDA)
        compatible version of the ietf-interfaces module
        is required.";
   }

   include ietf-ipv6-router-advertisements {
     revision-date 2018-01-25;
   }

   organization
     "IETF NETMOD - Networking Modeling Working Group";
   contact
     "WG Web:   <http://tools.ietf.org/wg/netmod/>
      WG List:  <mailto:rtgwg@ietf.org>

      Editor:   Ladislav Lhotka
                <mailto:lhotka@nic.cz>
                Acee Lindem
                <mailto:acee@cisco.com>
                Yingzhen Qu
                <mailto:yingzhen.qu@huawei.com>";

   description
     "This YANG module augments the 'ietf-routing' module with basic
      parameters for IPv6 unicast routing. The model fully conforms
      to 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 RFC XXXX; see
        the RFC itself for full legal notices.";
     reference "RFC XXXX";

     revision 2018-01-25 {
       description
         "Network Management Datastore Architecture (NMDA) revision";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management
          (NMDA Version)";
     }

     /* Identities */

     revision 2016-11-04 {
          description
            "Initial revision.";
          reference
            "RFC 8022: A YANG Data Model for Routing Management";
     }

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

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

     augment "/rt:routing/rt:ribs/rt:rib/rt:routes/rt:route/"
           + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
       when "derived-from-or-self(../../../rt:address-family, "
          + "'v6ur:ipv6-unicast')" {
```

```
         description
           "This augment is valid only for IPv6 unicast.";
       }
       description
         "Augment 'simple-next-hop' case in IPv6 unicast routes.";
       leaf next-hop-address {
         type inet:ipv6-address;
         description
           "IPv6 address of the next hop.";
       }
     }

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

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

     augment "/rt:routing/rt:ribs/rt:rib/rt:active-route/"
```

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

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

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

```
            "IPv6 address of the next hop.";
        }
      }

      /* Data node augmentations */

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

```
                routes.";
             leaf next-hop-address {
               type inet:ipv6-address;
               description
                 "IPv6 address of the next hop.";
             }
           }
         }
       }
     }
   }
 }

 /*
  * The subsequent data nodes are obviated and obsoleted by the
  * "Network Management Architecture" as described in
  * draft-ietf-netmod-revised-datastores.
  */
 augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route" {
   when "derived-from-or-self(../../rt:address-family,
           'v6ur:ipv6-unicast')" {
     description
       "This augment is valid only for IPv6 unicast.";
   }
   status obsolete;
   description
     "This leaf augments an IPv6 unicast route.";
   leaf destination-prefix {
     type inet:ipv6-prefix;
     status obsolete;
     description
       "IPv6 destination prefix.";
   }
 }
 augment "/rt:routing-state/rt:ribs/rt:rib/rt:routes/rt:route/"
         + "rt:next-hop/rt:next-hop-options/rt:simple-next-hop" {
   when "derived-from-or-self(../../../rt:address-family,
           'v6ur:ipv6-unicast')" {
     description
       "This augment is valid only for IPv6 unicast.";
   }
   status obsolete;
   description
     "Augment 'simple-next-hop' case in IPv6 unicast routes.";
   leaf next-hop-address {
     type inet:ipv6-address;
     status obsolete;
     description
```

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

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

```
   }
   <CODE ENDS>
```

9.1. IPv6 Router Advertisements Submodule

```
   <CODE BEGINS> file "ietf-ipv6-router-advertisements@2018-01-25.yang"
   submodule ietf-ipv6-router-advertisements {
     yang-version "1.1";

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

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

     import ietf-interfaces {
       prefix "if";
       description
         "A Network Management Datastore Architecture (NMDA)
          compatible version of the ietf-interfaces module
          is required.";
     }

     import ietf-ip {
       prefix "ip";
       description
         "A Network Management Datastore Architecture (NMDA)
          compatible version of the ietf-ip module is
          required.";
     }

     organization
       "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
     contact
       "WG Web:   <http://tools.ietf.org/wg/netmod/>
        WG List:  <mailto:rtgwg@ietf.org>

        Editor:   Ladislav Lhotka
                  <mailto:lhotka@nic.cz>
                  Acee Lindem
                  <mailto:acee@cisco.com>
                  Yingzhen Qu
                  <mailto:yingzhen.qu@huawei.com>";

     description
       "This YANG module augments the 'ietf-ip' module with
```

```
        parameters for IPv6 router advertisements. The model fully
        conforms to 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 RFC XXXX; see
        the RFC itself for full legal notices.";
     reference
       "RFC 4861: Neighbor Discovery for IP version 6 (IPv6).";

     revision 2018-01-25 {
       description
         "Network Management Datastore Architecture (NMDA) Revision";
       reference
         "RFC XXXX: A YANG Data Model for Routing Management
          (NMDA Version)";
     }

     revision 2016-11-04 {
          description
            "Initial revision.";
          reference
            "RFC 8022: A YANG Data Model for Routing Management";
     }

     augment "/if:interfaces/if:interface/ip:ipv6" {
       description
         "Augment interface configuration with parameters of IPv6
          router advertisements.";
       container ipv6-router-advertisements {
         description
           "Support for IPv6 Router Advertisements.";
         leaf send-advertisements {
           type boolean;
           default "false";
           description
             "A flag indicating whether or not the router sends
              periodic Router Advertisements and responds to
              Router Solicitations.";
```

```
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvSendAdvertisements.";
        }
        leaf max-rtr-adv-interval {
          type uint16 {
            range "4..65535";
          }
          units "seconds";
          default "600";
          description
            "The maximum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             MaxRtrAdvInterval.";
        }
        leaf min-rtr-adv-interval {
          type uint16 {
            range "3..1350";
          }
          units "seconds";
          must ". <= 0.75 * ../max-rtr-adv-interval" {
            description
              "The value MUST NOT be greater than 75% of
               'max-rtr-adv-interval'.";
          }
          description
            "The minimum time allowed between sending unsolicited
             multicast Router Advertisements from the interface.

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

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

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

```
              Advertisement.";
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvManagedFlag.";
         }
         leaf other-config-flag {
           type boolean;
           default "false";
           description
             "The value to be placed in the 'Other configuration'
              flag field in the Router Advertisement.";
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvOtherConfigFlag.";
         }
         leaf link-mtu {
           type uint32;
           default "0";
           description
             "The value to be placed in MTU options sent by the
              router. A value of zero indicates that no MTU options
              are sent.";
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvLinkMTU.";
         }
         leaf reachable-time {
           type uint32 {
             range "0..3600000";
           }
           units "milliseconds";
           default "0";
           description
             "The value to be placed in the Reachable Time field in
              the Router Advertisement messages sent by the router.
              A value of zero means unspecified (by this router).";
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvReachableTime.";
         }
         leaf retrans-timer {
           type uint32;
           units "milliseconds";
           default "0";
           description
             "The value to be placed in the Retrans Timer field in
              the Router Advertisement messages sent by the router.
              A value of zero means unspecified (by this router).";
```

```
           reference
             "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
              AdvRetransTimer.";
         }
         leaf cur-hop-limit {
           type uint8;
           description
             "The value to be placed in the Cur Hop Limit field in
              the Router Advertisement messages sent by the router.
              A value of zero means unspecified (by this router).

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

              IANA: IP Parameters,
              http://www.iana.org/assignments/ip-parameters";
         }
         leaf default-lifetime {
           type uint16 {
             range "0..65535";
           }
           units "seconds";
           description
             "The value to be placed in the Router Lifetime field of
              Router Advertisements sent from the interface, in
              seconds. It MUST be either zero or between
              max-rtr-adv-interval and 9000 seconds.  A value of zero
              default indicates that the router is not to be used as
              a router.  These limits may be overridden by specific
              documents that describe how IPv6 operates over
              different link layers.

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

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

               The link-local prefix SHOULD NOT be included in the
               list of advertised prefixes.";
          reference
            "RFC 4861: Neighbor Discovery for IP version 6 (IPv6) -
             AdvPrefixList.";
          list prefix {
            key "prefix-spec";
            description
              "Support for an advertised prefix entry.";
            leaf prefix-spec {
              type inet:ipv6-prefix;
              description
                "IPv6 address prefix.";
            }
            choice control-adv-prefixes {
              default "advertise";
              description
                "Either the prefix is explicitly removed from the
                 set of advertised prefixes, or the parameters with
                 which it is advertised are specified (default
                 case).";
              leaf no-advertise {
                type empty;
                description
                  "The prefix will not be advertised.

                   This can be used for removing the prefix from
                   the default set of advertised prefixes.";
              }
              case advertise {
                leaf valid-lifetime {
                  type uint32;
                  units "seconds";
                  default "2592000";
                  description
                    "The value to be placed in the Valid Lifetime
                     in the Prefix Information option.  The
                     designated value of all 1's (0xffffffff)
                      represents infinity.";
                  reference
                    "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvValidLifetime.";
                }
                leaf on-link-flag {
```

```
                  type boolean;
                  default "true";
                  description
                    "The value to be placed in the on-link flag
                     ('L-bit') field in the Prefix Information
                     option.";
                  reference
                    "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvOnLinkFlag.";
                }
                leaf preferred-lifetime {
                  type uint32;
                  units "seconds";
                  must ". <= ../valid-lifetime" {
                    description
                      "This value MUST NOT be greater than
                       valid-lifetime.";
                  }
                  default "604800";
                  description
                    "The value to be placed in the Preferred
                     Lifetime in the Prefix Information option.
                     The designated value of all 1's (0xffffffff)
                     represents infinity.";
                  reference
                    "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvPreferredLifetime.";
                }
                leaf autonomous-flag {
                  type boolean;
                  default "true";
                  description
                    "The value to be placed in the Autonomous Flag
                     field in the Prefix Information option.";
                  reference
                    "RFC 4861: Neighbor Discovery for IP version 6
                     (IPv6) - AdvAutonomousFlag.";
                }
              }
            }
          }
        }
      }
    }

    /*
     * The subsequent data nodes are obviated and obsoleted by the
     * "Network Management Architecture" as described in
```

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

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

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

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

                 An implementation SHOULD keep this value constant in
                 consecutive advertisements except when it is
                 explicitly changed in configuration.";
            }
            leaf on-link-flag {
              type boolean;
              status obsolete;
              description
                "The value that is placed in the on-link flag ('L-bit')
                 field in the Prefix Information option.";
            }
            leaf preferred-lifetime {
              type uint32;
```

```
               units "seconds";
               status obsolete;
               description
                 "The value that is placed in the Preferred Lifetime in
                  the Prefix Information option, in seconds.  The
                  designated value of all 1's (0xffffffff) represents
                  infinity.

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

10. IANA Considerations

[RFC8022] registered the following namespace URIs in the "IETF XML Registry" [RFC3688]:

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

URI: urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

[RFC8022] registered the following YANG modules in the "YANG Module Names" registry [RFC6020]:

```
Name:         ietf-routing
Namespace:    urn:ietf:params:xml:ns:yang:ietf-routing
Prefix:       rt
Reference:    RFC 8022

Name:         ietf-ipv4-unicast-routing
Namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing
Prefix:       v4ur
Reference:    RFC 8022

Name:         ietf-ipv6-unicast-routing
Namespace:    urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing
Prefix:       v6ur
Reference:    RFC 8022
```

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

```
Name:         ietf-ipv6-router-advertisements
Module:       ietf-ipv6-unicast-routing
Reference:    RFC 8022
```

11. Security Considerations

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

The NETCONF access control model [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.

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:

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

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

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

/routing/control-plane-protocols/control-plane-protocol: This list specifies the control-plane protocols configured on a device. Refer to the control plane models for a list of sensitive information.

/routing/ribs/rib: This list specifies the RIB and their contents for the device. Access to this information may disclose the network topology and or other information.

12. References

12.1. Normative References

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

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

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

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

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

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

[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>.

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

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

[I-D.ietf-netmod-rfc7223bis] Bjorklund, M., "A YANG Data Model for Interface Management", draft-ietf-netmod-rfc7223bis-03 (work in progress), January 2018.

[I-D.ietf-netmod-rfc7277bis] Bjorklund, M., "A YANG Data Model for IP Management", draft-ietf-netmod-rfc7277bis-03 (work in progress), January 2018.

[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>.

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <https://www.rfc-editor.org/info/rfc8022>.

[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>.

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

12.2. Informative References

[I-D.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-16 (work in progress), January 2018.

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

[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>.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-05 (work in progress), January 2018.

Appendix A. The Complete Schema Tree

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

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

```
     |  |                 +--rw (v6ur:next-hop-options)
     |  |                    +--:(v6ur:simple-next-hop)
     |  |                    |  +--rw v6ur:outgoing-interface?
     |  |                    |  |       if:interface-ref
     |  |                    |  +--rw v6ur:next-hop-address?
     |  |                    |          inet:ipv6-address
     |  |                    +--:(v6ur:special-next-hop)
     |  |                    |  +--rw v6ur:special-next-hop?
     |  |                    |          enumeration
     |  |                    +--:(v6ur:next-hop-list)
     |  |                       +--rw v6ur:next-hop-list
     |  |                          +--rw v6ur:next-hop* [index]
     |  |                             +--rw v6ur:index
     |  |                             |       string
     |  |                             +--rw v6ur:outgoing-interface?
     |  |                             |       if:interface-ref
     |  |                             +--rw v6ur:next-hop-address?
     |  |                                     inet:ipv6-address
     |  +--rw ribs
     |     +--rw rib* [name]
     |        +--rw name              string
     |        +--rw address-family    identityref
     |        +--ro default-rib?      boolean {multiple-ribs}?
     |        +--ro routes
     |        |  +--ro route*
     |        |     +--ro route-preference?          route-preference
     |        |     +--ro next-hop
     |        |     |  +--ro (next-hop-options)
     |        |     |     +--:(simple-next-hop)
     |        |     |     |  +--ro outgoing-interface?
     |        |     |     |  |       if:interface-ref
     |        |     |     |  +--ro v4ur:next-hop-address?
     |        |     |     |  |       inet:ipv4-address
     |        |     |     |  +--ro v6ur:next-hop-address?
     |        |     |     |          inet:ipv6-address
     |        |     |     +--:(special-next-hop)
     |        |     |     |  +--ro special-next-hop?        enumeration
     |        |     |     +--:(next-hop-list)
     |        |     |        +--ro next-hop-list
     |        |     |           +--ro next-hop*
     |        |     |              +--ro outgoing-interface?
     |        |     |              |       if:interface-ref
     |        |     |              +--ro v4ur:address?
     |        |     |              |       inet:ipv4-address
     |        |     |              +--ro v6ur:address?
     |        |     |                      inet:ipv6-address
     |        |     +--ro source-protocol            identityref
     |        |     +--ro active?                    empty
```

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

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

```
                       |     |  o--ro special-next-hop?
                       |     |          enumeration
                       |     o--:(next-hop-list)
                       |        o--ro next-hop-list
                       |           o--ro next-hop*
                       |              o--ro outgoing-interface?
                       |              |       if:interface-ref
                       |              o--ro v4ur:next-hop-address?
                       |              |       inet:ipv4-address
                       |              o--ro v6ur:next-hop-address?
                       |                      inet:ipv6-address
                       o--ro source-protocol            identityref
                       o--ro active?                    empty
                       o--ro last-updated?
                       |       yang:date-and-time
                       o--ro v4ur:destination-prefix?
                       |       inet:ipv4-prefix
                       o--ro v6ur:destination-prefix?
                               inet:ipv6-prefix
   module: ietf-ipv6-unicast-routing
     augment /if:interfaces/if:interface/ip:ipv6:
       +--rw ipv6-router-advertisements
          +--rw send-advertisements?    boolean
          +--rw max-rtr-adv-interval?   uint16
          +--rw min-rtr-adv-interval?   uint16
          +--rw managed-flag?           boolean
          +--rw other-config-flag?      boolean
          +--rw link-mtu?               uint32
          +--rw reachable-time?         uint32
          +--rw retrans-timer?          uint32
          +--rw cur-hop-limit?          uint8
          +--rw default-lifetime?       uint16
          +--rw prefix-list
             +--rw prefix* [prefix-spec]
                +--rw prefix-spec           inet:ipv6-prefix
                +--rw (control-adv-prefixes)?
                   +--:(no-advertise)
                   |  +--rw no-advertise?         empty
                   +--:(advertise)
                      +--rw valid-lifetime?       uint32
                      +--rw on-link-flag?         boolean
                      +--rw preferred-lifetime?   uint32
                      +--rw autonomous-flag?      boolean
     augment /if:interfaces-state/if:interface/ip:ipv6:
       o--ro ipv6-router-advertisements
          o--ro send-advertisements?    boolean
          o--ro max-rtr-adv-interval?   uint16
          o--ro min-rtr-adv-interval?   uint16
```

```
         o--ro managed-flag?           boolean
         o--ro other-config-flag?      boolean
         o--ro link-mtu?               uint32
         o--ro reachable-time?         uint32
         o--ro retrans-timer?          uint32
         o--ro cur-hop-limit?          uint8
         o--ro default-lifetime?       uint16
         o--ro prefix-list
            o--ro prefix* [prefix-spec]
               o--ro prefix-spec           inet:ipv6-prefix
               o--ro valid-lifetime?       uint32
               o--ro on-link-flag?         boolean
               o--ro preferred-lifetime?   uint32
               o--ro autonomous-flag?      boolean
```

Appendix B. Minimum Implementation

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

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

In addition to the mandatory instance of the "direct" pseudo-protocol, a minimum implementation should support configuring instance(s) of the "static" pseudo-protocol.

For hosts that are never intended to act as routers, the ability to turn on sending IPv6 router advertisements (Section 5.4) should be removed.

Platforms with severely constrained resources may use deviations for restricting the data model, e.g., limiting the number of "static" control-plane protocol instances.

Appendix C. Example: Adding a New Control-Plane Protocol

This appendix demonstrates how the core routing data model can be extended to support a new control-plane protocol. The YANG module "example-rip" shown below is intended as an illustration rather than a real definition of a data model for the Routing Information Protocol (RIP). For the sake of brevity, this module does not obey all the guidelines specified in [I-D.ietf-netmod-rfc6087bis]. See also Section 5.3.2.

```
module example-rip {

  yang-version "1.1";

  namespace "http://example.com/rip";

  prefix "rip";

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-routing {
    prefix "rt";
  }

  identity rip {
    base rt:routing-protocol;
    description
      "Identity for the Routing Information Protocol (RIP).";
  }

  typedef rip-metric {
    type uint8 {
      range "0..16";
    }
  }

  grouping route-content {
    description
      "This grouping defines RIP-specific route attributes.";
    leaf metric {
      type rip-metric;
    }
    leaf tag {
      type uint16;
      default "0";
      description
        "This leaf may be used to carry additional info, e.g.,
         autonomous system (AS) number.";
    }
  }

  augment "/rt:routing/rt:ribs/rt:rib/rt:routes/rt:route" {
    when "derived-from-or-self(rt:source-protocol, 'rip:rip')" {
```

```
         description
           "This augment is only valid for a route whose source
            protocol is RIP.";
       }
       description
         "RIP-specific route attributes.";
       uses route-content;
     }

     augment "/rt:routing/rt:ribs/rt:rib/rt:active-route/"
           + "rt:output/rt:route" {
       description
         "RIP-specific route attributes in the output of 'active-route'
          RPC.";
       uses route-content;
     }

     augment "/rt:routing/rt:control-plane-protocols/"
           + "rt:control-plane-protocol" {
       when "derived-from-or-self(rt:type,'rip:rip')" {
         description
           "This augment is only valid for a routing protocol instance
            of type 'rip'.";
       }
       container rip {
         presence "RIP configuration";
         description
           "RIP instance configuration.";
         container interfaces {
           description
             "Per-interface RIP configuration.";
           list interface {
             key "name";
             description
               "RIP is enabled on interfaces that have an entry in this
                list, unless 'enabled' is set to 'false' for that
                entry.";
             leaf name {
               type if:interface-ref;
             }
             leaf enabled {
               type boolean;
               default "true";
             }
             leaf metric {
               type rip-metric;
               default "1";
             }
```

```
          }
        }
        leaf update-interval {
          type uint8 {
            range "10..60";
          }
          units "seconds";
          default "30";
          description
            "Time interval between periodic updates.";
        }
      }
    }
  }
```

Appendix D. Data Tree Example

This section contains an example of an instance data tree from the operational state, in the JSON encoding [RFC7951]. The data conforms to a data model that is defined by the following YANG library specification [RFC7895]:


```
 {
   "ietf-yang-library:modules-state": {
     "module-set-id": "c2e1f54169aa7f36e1a6e8d0865d441d3600f9c4",
     "module": [
       {
         "name": "ietf-routing",
         "revision": "2018-01-25",
         "feature": [
           "multiple-ribs",
           "router-id"
         ],
         "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
         "conformance-type": "implement"
       },
       {
         "name": "ietf-ipv4-unicast-routing",
         "revision": "2018-01-25",
         "namespace":
           "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
         "conformance-type": "implement"
       },
       {
         "name": "ietf-ipv6-unicast-routing",
         "revision": "2018-01-25",
         "namespace":
           "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
```

```
            "conformance-type": "implement",
            "submodule": [
              {
                "name": "ietf-ipv6-router-advertisements",
                "revision": "2018-01-25"
              }
            ]
          },
          {
            "name": "ietf-interfaces",
            "revision": "2017-12-16",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-inet-types",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
            "revision": "2013-07-15",
            "conformance-type": "import"
          },
          {
            "name": "ietf-yang-types",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
            "revision": "2013-07-15",
            "conformance-type": "import"
          },
          {
            "name": "iana-if-type",
            "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
            "revision": "2014-05-08",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-ip",
            "revision": "2017-12-16",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
            "conformance-type": "implement"
          }
        ]
      }
    }
```


A simple network setup as shown in Figure 2 is assumed: router "A" uses static default routes with the "ISP" router as the next hop. IPv6 router advertisements are configured only on the "eth1" interface and disabled on the upstream "eth0" interface.

```
                  +-----------------+
                  |                 |
                  |    Router ISP   |
                  |                 |
                  +--------+--------+
                           |2001:db8:0:1::2
                           |192.0.2.2
                           |
                           |
                           |2001:db8:0:1::1
                       eth0|192.0.2.1
                  +--------+--------+
                  |                 |
                  |     Router A    |
                  |                 |
                  +--------+--------+
                       eth1|198.51.100.1
                           |2001:db8:0:2::1
                           |
```

Figure 2: Example of Network Configuration

The instance data tree could then be as follows:


```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "description": "Uplink to ISP.",
        "phys-address": "00:0C:42:E5:B1:E9",
        "oper-status": "up",
        "statistics": {
          "discontinuity-time": "2015-10-24T17:11:27+02:00"
        },
        "ietf-ip:ipv4": {
          "forwarding": true,
          "mtu": 1500,
          "address": [
            {
              "ip": "192.0.2.1",
              "prefix-length": 24
            }
          ]
        },
        "ietf-ip:ipv6": {
          "forwarding": true,
```

```
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:1::1",
                "prefix-length": 64
              }
            ],
            "autoconf": {
              "create-global-addresses": false
            },
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": false
            }
          }
        },
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "description": "Interface to the internal network.",
          "phys-address": "00:0C:42:E5:B1:EA",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2015-10-24T17:11:29+02:00"
          },
          "ietf-ip:ipv4": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "198.51.100.1",
                "prefix-length": 24
              }
            ]
          },
          "ietf-ip:ipv6": {
            "forwarding": true,
            "mtu": 1500,
            "address": [
              {
                "ip": "2001:0db8:0:2::1",
                "prefix-length": 64
              }
            ],
            "autoconf": {
              "create-global-addresses": false
            },
            "ietf-ipv6-unicast-routing:ipv6-router-advertisements": {
              "send-advertisements": true,
```

```
              "prefix-list": {
                "prefix": [
                  {
                    "prefix-spec": "2001:db8:0:2::/64"
                  }
                ]
              }
            }
          }
        }
      ]
    },

    "ietf-routing:routing": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:static",
            "name": "st0",
            "description":
              "Static routing is used for the internal network.",
            "static-routes": {
              "ietf-ipv4-unicast-routing:ipv4": {
                "route": [
                  {
                    "destination-prefix": "0.0.0.0/0",
                    "next-hop": {
                      "next-hop-address": "192.0.2.2"
                    }
                  }
                ]
              },
              "ietf-ipv6-unicast-routing:ipv6": {
                "route": [
                  {
                    "destination-prefix": "::/0",
                    "next-hop": {
                      "next-hop-address": "2001:db8:0:1::2"
                    }
                  }
                ]
              }
            }
          }
        ]
      },
      "ribs": {
```

```
         "rib": [
           {
             "name": "ipv4-master",
             "address-family":
               "ietf-ipv4-unicast-routing:ipv4-unicast",
             "default-rib": true,
             "routes": {
               "route": [
                 {
                   "ietf-ipv4-unicast-routing:destination-prefix":
                     "192.0.2.1/24",
                   "next-hop": {
                     "outgoing-interface": "eth0"
                   },
                   "route-preference": 0,
                   "source-protocol": "ietf-routing:direct",
                   "last-updated": "2015-10-24T17:11:27+02:00"
                 },
                 {
                   "ietf-ipv4-unicast-routing:destination-prefix":
                     "198.51.100.0/24",
                   "next-hop": {
                     "outgoing-interface": "eth1"
                   },
                   "source-protocol": "ietf-routing:direct",
                   "route-preference": 0,
                   "last-updated": "2015-10-24T17:11:27+02:00"
                 },
                 {
                   "ietf-ipv4-unicast-routing:destination-prefix":
                     "0.0.0.0/0",
                   "source-protocol": "ietf-routing:static",
                   "route-preference": 5,
                   "next-hop": {
                     "ietf-ipv4-unicast-routing:next-hop-address":
                       "192.0.2.2"
                   },
                   "last-updated": "2015-10-24T18:02:45+02:00"
                 }
               ]
             }
           },
           {
             "name": "ipv6-master",
             "address-family":
               "ietf-ipv6-unicast-routing:ipv6-unicast",
             "default-rib": true,
             "routes": {
```

```
               "route": [
                 {
                   "ietf-ipv6-unicast-routing:destination-prefix":
                     "2001:db8:0:1::/64",
                   "next-hop": {
                     "outgoing-interface": "eth0"
                   },
                   "source-protocol": "ietf-routing:direct",
                   "route-preference": 0,
                   "last-updated": "2015-10-24T17:11:27+02:00"
                 },
                 {
                   "ietf-ipv6-unicast-routing:destination-prefix":
                     "2001:db8:0:2::/64",
                   "next-hop": {
                     "outgoing-interface": "eth1"
                   },
                   "source-protocol": "ietf-routing:direct",
                   "route-preference": 0,
                   "last-updated": "2015-10-24T17:11:27+02:00"
                 },
                 {
                   "ietf-ipv6-unicast-routing:destination-prefix":
                     "::/0",
                   "next-hop": {
                     "ietf-ipv6-unicast-routing:next-hop-address":
                       "2001:db8:0:1::2"
                   },
                   "source-protocol": "ietf-routing:static",
                   "route-preference": 5,
                   "last-updated": "2015-10-24T18:02:45+02:00"
                 }
               ]
             }
           }
         ]
       }
     }
   }
```


Appendix E. NETCONF Get Data Reply Example

This section gives an example of an XML reply to the NETCONF <get-data> request for <operational> for a device that implements the example data models above.


```
<rpc-reply
```

```
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    message-id="101">
    <data>
      <routing
        xmlns="urn:ietf:params:xml:ns:yang:ietf-routing"
        xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

        <router-id or:origin="or:intended">192.0.2.1</router-id>
        <control-plane-protocols or:origin="or:intended">
          <control-plane-protocol>
            <type>ietf-routing:static</type>
            <name></name>
            <static-routes>
              <ietf-ipv4-unicast-routing:ipv4>
                <route>
                  <destination-prefix>0.0.0.0/0</destination-prefix>
                  <next-hop>
                    <next-hop-address>192.0.2.2</next-hop-address>
                  </next-hop>
                </route>
              </ietf-ipv4-unicast-routing:ipv4>
              <ietf-ipv6-unicast-routing:ipv6>
                <route>
                  <destination-prefix>::/0</destination-prefix>
                  <next-hop>
                    <next-hop-address>2001:db8:0:1::2</next-hop-address>
                  </next-hop>
                </route>
              </ietf-ipv6-unicast-routing:ipv6>
            </static-routes>
          </control-plane-protocol>
        </control-plane-protocols>

        <ribs>
          <rib or:origin="or:intended">
            <name>ipv4-master</name>
            <address-family>
              ietf-ipv4-unicast-routing:ipv4-unicast
            </address-family>
            <default-rib>true</default-rib>
            <routes>
              <route>
                <ietf-ipv4-unicast-routing:destination-prefix>
                  192.0.2.1/24
                </ietf-ipv4-unicast-routing:destination-prefix>
                <next-hop>
                  <outgoing-interface>eth0</outgoing-interface>
                </next-hop>
```

```
                <route-preference>0</route-preference>
                <source-protocol>ietf-routing:direct</source-protocol>
                <last-updated>2015-10-24T17:11:27+02:00</last-updated>
              </route>
              <route>
                <ietf-ipv4-unicast-routing:destination-prefix>
                  198.51.100.0/24
                </ietf-ipv4-unicast-routing:destination-prefix>
                <next-hop>
                  <outgoing-interface>eth1</outgoing-interface>
                </next-hop>
                <route-preference>0</route-preference>
                <source-protocol>ietf-routing:direct</source-protocol>
                <last-updated>2015-10-24T17:11:27+02:00</last-updated>
              </route>
              <route>
                <ietf-ipv4-unicast-routing:destination-prefix>0.0.0.0/0
                </ietf-ipv4-unicast-routing:destination-prefix>
                <next-hop>
                  <ietf-ipv4-unicast-routing:next-hop-address>192.0.2.2
                  </ietf-ipv4-unicast-routing:next-hop-address>
                </next-hop>
                <route-preference>5</route-preference>
                <source-protocol>ietf-routing:static</source-protocol>
                <last-updated>2015-10-24T18:02:45+02:00</last-updated>
              </route>
            </routes>
          </rib>
          <rib or:origin="or:intended">
            <name>ipv6-master</name>
            <address-family>
              ietf-ipv6-unicast-routing:ipv6-unicast
            </address-family>
            <default-rib>true</default-rib>
            <routes>
              <route>
                <ietf-ipv6-unicast-routing:destination-prefix>
                  2001:db8:0:1::/64
                </ietf-ipv6-unicast-routing:destination-prefix>
                <next-hop>
                  <outgoing-interface>eth0</outgoing-interface>
                </next-hop>
                <route-preference>0</route-preference>
                <source-protocol>ietf-routing:direct</source-protocol>
                <last-updated>2015-10-24T17:11:27+02:00</last-updated>
              </route>
              <route>
                <ietf-ipv6-unicast-routing:destination-prefix>
```

```
              2001:db8:0:2::/64
            </ietf-ipv6-unicast-routing:destination-prefix>
            <next-hop>
              <outgoing-interface>eth1</outgoing-interface>
            </next-hop>
            <route-preference>0</route-preference>
            <source-protocol>ietf-routing:direct</source-protocol>
            <last-updated>2015-10-24T17:11:27+02:00</last-updated>
          </route>
          <route>
            <ietf-ipv6-unicast-routing:destination-prefix>::/0
            </ietf-ipv6-unicast-routing:destination-prefix>
            <next-hop>
              <ietf-ipv6-unicast-routing:next-hop-address>
                2001:db8:0:1::2
              </ietf-ipv6-unicast-routing:next-hop-address>
            </next-hop>
            <route-preference>5</route-preference>
            <source-protocol>ietf-routing:static</source-protocol>
            <last-updated>2015-10-24T18:02:45+02:00</last-updated>
          </route>
        </routes>
      </rib>
    </ribs>
  </routing>
 </data>
</rpc-reply>
```


Acknowledgments

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

Authors' Addresses

Ladislav Lhotka
CZ.NIC

EMail: lhotka@nic.cz

Acee Lindem
Cisco Systems

EMail: acee@cisco.com

Yingzhen Qu
Huawei
2330 Central Expressway
Santa Clara CA 95050
USA

EMail: yingzhen.qu@huawei.com

Network Working Group M. Bjorklund
Internet-Draft Tail-f Systems
Intended status: Standards Track L. Lhotka
Expires: April 24, 2018 CZ.NIC
October 21, 2017

YANG Schema Mount
draft-ietf-netmod-schema-mount-08

Abstract

This document defines a mechanism to combine YANG modules into the schema defined in other YANG modules.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Modularity and extensibility were among the leading design principles of the YANG data modeling language. As a result, the same YANG module can be combined with various sets of other modules and thus form a data model that is tailored to meet the requirements of a specific use case. Server implementors are only required to specify all YANG modules comprising the data model (together with their revisions and other optional choices) in the YANG library data ([RFC7895], and Section 5.6.4 of [RFC7950]) implemented by the server. Such YANG modules appear in the data model "side by side", i.e., top-level data nodes of each module - if there are any - are also top-level nodes of the overall data model.

Furthermore, YANG has two mechanisms for contributing a schema hierarchy defined elsewhere to the contents of an internal node of the schema tree; these mechanisms are realized through the following YANG statements:

o The "uses" statement explicitly incorporates the contents of a grouping defined in the same or another module. See Section 4.2.6 of [RFC7950] for more details.

o The "augment" statement explicitly adds contents to a target node defined in the same or another module. See Section 4.2.8 of [RFC7950] for more details.

With both mechanisms, the source or target YANG module explicitly defines the exact location in the schema tree where the new nodes are placed.

In some cases these mechanisms are not sufficient; it is often necessary that an existing module (or a set of modules) is added to the data model starting at a non-root location. For example, YANG modules such as "ietf-interfaces" [RFC7223] are often defined so as to be used in a data model of a physical device. Now suppose we want to model a device that supports multiple logical devices [I-D.ietf-rtgwg-lne-model], each of which has its own instantiation of "ietf-interfaces", and possibly other modules, but, at the same time, we want to be able to manage all these logical devices from the master device. Hence, we would like to have a schema like this:

```
+--rw interfaces
|  +--rw interface* [name]
|     ...
+--rw logical-device* [name]
   +--rw name
   |   ...
   +--rw interfaces
     +--rw interface* [name]
        ...
```

With the "uses" approach, the complete schema tree of "ietf-interfaces" would have to be wrapped in a grouping, and then this grouping would have to be used at the top level (for the master device) and then also in the "logical-device" list (for the logical devices). This approach has several disadvantages:

o It is not scalable because every time there is a new YANG module that needs to be added to the logical device model, we have to update the model for logical devices with another "uses" statement pulling in contents of the new module.

o Absolute references to nodes defined inside a grouping may break if the grouping is used in different locations.

- o Nodes defined inside a grouping belong to the namespace of the module where it is used, which makes references to such nodes from other modules difficult or even impossible.

- o It would be difficult for vendors to add proprietary modules when the "uses" statements are defined in a standard module.

With the "augment" approach, "ietf-interfaces" would have to augment the "logical-device" list with all its nodes, and at the same time define all its nodes at the top level. The same hierarchy of nodes would thus have to be defined twice, which is clearly not scalable either.

This document introduces a new generic mechanism, denoted as schema mount, that allows for mounting one data model consisting of any number of YANG modules at a specified location of another (parent) schema. Unlike the "uses" and "augment" approaches discussed above, the mounted modules needn't be specially prepared for mounting and, consequently, existing modules such as "ietf-interfaces" can be mounted without any modifications.

The basic idea of schema mount is to label a data node in the parent schema as the mount point, and then define a complete data model to be attached to the mount point so that the labeled data node effectively becomes the root node of the mounted data model.

In principle, the mounted schema can be specified at three different phases of the data model life cycle:

1. Design-time: the mounted schema is defined along with the mount point in the parent YANG module. In this case, the mounted schema has to be the same for every implementation of the parent module.

2. Implementation-time: the mounted schema is defined by a server implementor and is as stable as YANG library information, i.e., it may change after an upgrade of server software but not after rebooting the server. Also, a client can learn the entire schema together with YANG library data.

3. Run-time: the mounted schema is defined by instance data that is part of the mounted data model. If there are multiple instances of the same mount point (e.g., in multiple entries of a list), the mounted data model may be different for each instance.

The schema mount mechanism defined in this document provides support only for the latter two cases. Design-time mounts are outside the scope of this document, and could be possibly dealt with in a future revision of the YANG data modeling language.

Schema mount applies to the data model, and specifically does not assume anything about the source of instance data for the mounted schemas. It may be implemented using the same instrumentation as the rest of the system, or it may be implemented by querying some other system. Future specifications may define mechanisms to control or monitor the implementation of specific mount points.

This document allows mounting of complete data models only. Other specifications may extend this model by defining additional mechanisms such as mounting sub-hierarchies of a module.

2. Terminology and Notation

The keywords "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].

The following terms are defined in [RFC6241] and are not redefined here:

- client
- notification
- server

The following terms are defined in [RFC7950] and are not redefined here:

- action
- container
- list
- operation

The following terms are defined in [RFC7223] and are not redefined here:

- system-controlled interface

Tree diagrams used in this document follow the notation defined in [I-D.ietf-netmod-yang-tree-diagrams].

2.1. Glossary of New Terms

- o inline schema: a mounted schema whose definition is provided as part of the mounted data, using YANG library [RFC7895].

- o mount point: container or list node whose definition contains the "mount-point" extension statement. The argument of the "mount-point" statement defines a label for the mount point.

- o parent schema (of a particular mounted schema): the schema that contains the mount point for the mounted schema.

- o top-level schema: a schema according to [RFC7950] in which schema trees of each module (except augments) start at the root node.

2.2. Namespace Prefixes

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

Prefix	YANG module	Reference
yangmnt	ietf-yang-schema-mount	Section 8
inet	ietf-inet-types	[RFC6991]
yang	ietf-yang-types	[RFC6991]
yanglib	ietf-yang-library	[RFC7895]

Table 1: Namespace Prefixes

3. Schema Mount

The schema mount mechanism defined in this document provides a new extensibility mechanism for use with YANG 1.1. In contrast to the existing mechanisms described in Section 1, schema mount defines the relationship between the source and target YANG modules outside these modules. The procedure consists of two separate steps that are described in the following subsections.

3.1. Mount Point Definition

A "container" or "list" node becomes a mount point if the "mount-point" extension (defined in the "ietf-yang-schema-mount" module) is used in its definition. This extension can appear only as a substatement of "container" and "list" statements.

The argument of the "mount-point" extension is a YANG identifier that defines a label for the mount point. A module MAY contain multiple "mount-point" statements having the same argument.

It is therefore up to the designer of the parent schema to decide about the placement of mount points. A mount point can also be made conditional by placing "if-feature" and/or "when" as substatements of the "container" or "list" statement that represents the mount point.

The "mount-point" statement MUST NOT be used in a YANG version 1 module. Note, however, that modules written in any YANG version, including version 1, can be mounted under a mount point.

Note that the "mount-point" statement does not define a new data node.

3.2. Specification of the Mounted Schema

Mounted schemas for all mount points in the parent schema are determined from state data in the "yangmnt:schema-mounts" container. Data in this container is intended to be as stable as data in the top-level YANG library [RFC7895]. In particular, it SHOULD NOT change during the same management session.

Generally, the modules that are mounted under a mount point have no relation to the modules in the parent schema; specifically, if a module is mounted it may or may not be present in the parent schema and, if present, its data will generally have no relationship to the data of the parent. Exceptions are possible and such needs to be defined in the model defining the exception, e.g., the interface module in [I-D.ietf-rtgwg-lne-model].

The "schema-mounts" container has the "mount-point" list as one of its children. Every entry of this list refers through its key to a mount point and specifies the mounted schema.

If a mount point is defined in the parent schema but does not have an entry in the "mount-point" list, then the mounted schema is void, i.e., instances of that mount point MUST NOT contain any data above those that are defined in the parent schema.

If multiple mount points with the same name are defined in the same module - either directly or because the mount point is defined in a grouping and the grouping is used multiple times - then the corresponding "mount-point" entry applies equally to all such mount points.

The "config" property of mounted schema nodes is overridden and all nodes in the mounted schema are read-only ("config false") if at least one of the following conditions is satisfied for a mount point:

- o the mount point is itself defined as "config false"

- o the "config" leaf in the corresponding entry of the "mount-point" list is set to "false".

An entry of the "mount-point" list can specify the mounted schema in two different ways:

1. by stating that the schema is available inline, i.e., in run-time instance data; or

2. by referring to one or more entries of the "schema" list in the same instance of "schema-mounts".

In case 1, the mounted schema is determined at run time: every instance of the mount point that exists in the parent tree MUST contain a copy of YANG library data [RFC7895] that defines the mounted schema exactly as for a top-level data model. A client is expected to retrieve this data from the instance tree, possibly after creating the mount point. Instances of the same mount point MAY use different mounted schemas.

In case 2, the mounted schema is defined by the combination of all "schema" entries referred to in the "use-schema" list. In this case, the mounted schema is specified as implementation-time data that can be retrieved together with YANG library data for the parent schema, i.e., even before any instances of the mount point exist. However, the mounted schema has to be the same for all instances of the mount point. Note, that in this case a mount point may include a mounted YANG library module and the data contained in the mounted module MUST exactly match the data contained in the "schema" entries associated with the mount point.

Each entry of the "schema" list contains:

- o a list in the YANG library format specifying all YANG modules (and revisions etc.) that are implemented or imported in the mounted schema. Note that this includes modules that solely augment other listed modules;

- o (optionally) a new "mount-point" list that applies to mount points defined within the mounted schema.

3.3. Multiple Levels of Schema Mount

YANG modules in a mounted schema MAY again contain mount points under which subschemas can be mounted. Consequently, it is possible to construct data models with an arbitrary number of schema levels. A subschema for a mount point contained in a mounted module can be specified in one of the following ways:

- o by implementing "ietf-yang-library" and "ietf-yang-schema-mount" modules in the mounted schema, and specifying the subschemas exactly as it is done in the top-level schema

- o by using the "mount-point" list inside the corresponding "schema" entry.

The former method is applicable to both "inline" and "use-schema" cases whereas the latter requires the "use-schema" case. On the other hand, the latter method allows for a compact representation of a multi-level schema the does not rely on the presence of any instance data.

4. Referring to Data Nodes in the Parent Schema

A fundamental design principle of schema mount is that the mounted data model works exactly as a top-level data model, i.e., it is confined to the "mount jail". This means that all paths in the mounted data model (in leafrefs, instance-identifiers, XPath expressions, and target nodes of augments) are interpreted with the mount point as the root node. YANG modules of the mounted schema as well as corresponding instance data thus cannot refer to schema nodes or instance data outside the mount jail.

However, this restriction is sometimes too severe. A typical example is network instances (NI) [I-D.ietf-rtgwg-ni-model], where each NI has its own routing engine but the list of interfaces is global and shared by all NIs. If we want to model this organization with the NI schema mounted using schema mount, the overall schema tree would look schematically as follows:

```
     +--rw interfaces
     |  +--rw interface* [name]
     |     ...
     +--rw network-instances
        +--rw network-instance* [name]
           +--rw name
           +--rw root
              +--rw routing
                 ...
```

Here, the "root" node is the mount point for the NI schema. Routing configuration inside an NI often needs to refer to interfaces (at least those that are assigned to the NI), which is impossible unless such a reference can point to a node in the parent schema (interface name).

Therefore, schema mount also allows for such references. For every schema mounted using the "use-schema" method, it is possible to specify a leaf-list named "parent-reference" that contains zero or more XPath 1.0 expressions. Each expression is evaluated with the node in the parent data tree where the mount point is defined as the context node. The result of this evaluation MUST be a nodeset (see the description of the "parent-reference" node for a complete definition of the evaluation context). For the purposes of evaluating XPath expressions within the mounted data tree, the union of all such nodesets is added to the accessible data tree.

It is worth emphasizing that

o The nodes specified in "parent-reference" leaf-list are available in the mounted schema only for XPath evaluations. In particular, they cannot be accessed there via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040].

o The mechanism of referencing nodes in the parent schema is not available for schemas mounted using the "inline" method.

5. RPC operations and Notifications

If a mounted YANG module defines an RPC operation, clients can invoke this operation by representing it as an action defined for the corresponding mount point, see Section 7.15 of ^RFC7950. An example of this is given in Appendix A.4.

Similarly, if the server emits a notification defined at the top level of any mounted module, it MUST be represented as if the notification was connected to the mount point, see Section 7.16 of [RFC7950].

Note, inline actions and notifications will not work when they are contained within a list node without a "key" statement (see section 7.15 and 7.16 of [RFC7950]). Therefore, to be useful, mount points which contain modules with RPCs, actions, and notifications SHOULD NOT have any ancestor node that is a list node without a "key" statement. This requirement applies to the definition of modules using the "mount-point" extension statement.

6. Implementation Notes

Network management of devices that use a data model with schema mount can be implemented in different ways. However, the following implementations options are envisioned as typical:

- o shared management: instance data of both parent and mounted schemas are accessible within the same management session.

- o split management: one (master) management session has access to instance data of both parent and mounted schemas but, in addition, an extra session exists for every instance of the mount point, having access only to the mounted data tree.

7. Data Model

This document defines the YANG 1.1 module [RFC7950] "ietf-yang-schema-mount", which has the following structure:

```
   module: ietf-yang-schema-mount
       +--ro schema-mounts
          +--ro namespace* [prefix]
          |  +--ro prefix    yang:yang-identifier
          |  +--ro uri?      inet:uri
          +--ro mount-point* [module label]
          |  +--ro module        yang:yang-identifier
          |  +--ro label         yang:yang-identifier
          |  +--ro config?       boolean
          |  +--ro (schema-ref)
          |     +--:(inline)
          |     |  +--ro inline?       empty
          |     +--:(use-schema)
          |        +--ro use-schema* [name]
          |           +--ro name
          |           |       -> /schema-mounts/schema/name
          |           +--ro parent-reference*   yang:xpath1.0
          +--ro schema* [name]
             +--ro name           string
             +--ro module* [name revision]
             |  +--ro name                yang:yang-identifier
             |  +--ro revision            union
             |  +--ro schema?             inet:uri
             |  +--ro namespace           inet:uri
             |  +--ro feature*            yang:yang-identifier
             |  +--ro deviation* [name revision]
             |  |  +--ro name        yang:yang-identifier
             |  |  +--ro revision    union
             |  +--ro conformance-type    enumeration
             |  +--ro submodule* [name revision]
             |     +--ro name        yang:yang-identifier
             |     +--ro revision    union
             |     +--ro schema?     inet:uri
             +--ro mount-point* [module label]
                +--ro module        yang:yang-identifier
                +--ro label         yang:yang-identifier
                +--ro config?       boolean
                +--ro (schema-ref)
                   +--:(inline)
                   |  +--ro inline?       empty
                   +--:(use-schema)
                      +--ro use-schema* [name]
                         +--ro name
                         |       -> /schema-mounts/schema/name
                         +--ro parent-reference*   yang:xpath1.0
```

8. Schema Mount YANG Module

This module references [RFC6991] and [RFC7895].

```
<CODE BEGINS> file "ietf-yang-schema-mount@2017-10-09.yang"

module ietf-yang-schema-mount {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount";
  prefix yangmnt;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-yang-library {
    prefix yanglib;
    reference
      "RFC 7895: YANG Module Library";
  }

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

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

     Editor:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

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

  description
    "This module defines a YANG extension statement that can be used
     to incorporate data models defined in other YANG modules in a
     module. It also defines operational state data that specify the
     overall structure of the data model.
```

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

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

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

     revision 2017-10-09 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: YANG Schema Mount";
     }

     /*
      * Extensions
      */

     extension mount-point {
       argument label;
       description
         "The argument 'label' is a YANG identifier, i.e., it is of the
          type 'yang:yang-identifier'.

          The 'mount-point' statement MUST NOT be used in a YANG
          version 1 module, neither explicitly nor via a 'uses'
          statement.

          The 'mount-point' statement MAY be present as a substatement
          of 'container' and 'list', and MUST NOT be present elsewhere.
          There MUST NOT be more than one 'mount-point' statement in a
          given 'container' or 'list' statement.

          If a mount point is defined within a grouping, its label is
          bound to the module where the grouping is used.
```

```
           A mount point defines a place in the node hierarchy where
           other data models may be attached. A server that implements a
           module with a mount point populates the
           /schema-mounts/mount-point list with detailed information on
           which data models are mounted at each mount point.

           Note that the 'mount-point' statement does not define a new
           data node.";
      }

      /*
       * Groupings
       */

      grouping mount-point-list {
        description
          "This grouping is used inside the 'schema-mounts' container and
           inside the 'schema' list.";
        list mount-point {
          key "module label";
          description
            "Each entry of this list specifies a schema for a particular
             mount point.

             Each mount point MUST be defined using the 'mount-point'
             extension in one of the modules listed in the corresponding
             YANG library instance with conformance type 'implement'. The
             corresponding YANG library instance is:

             - standard YANG library state data as defined in RFC 7895,
               if the 'mount-point' list is a child of 'schema-mounts',

             - the contents of the sibling 'yanglib:modules-state'
               container, if the 'mount-point' list is a child of
               'schema'.";
          leaf module {
            type yang:yang-identifier;
            description
              "Name of a module containing the mount point.";
          }
          leaf label {
            type yang:yang-identifier;
            description
              "Label of the mount point defined using the 'mount-point'
               extension.";
          }
          leaf config {
            type boolean;
```

```
           default "true";
           description
             "If this leaf is set to 'false', then all data nodes in the
              mounted schema are read-only (config false), regardless of
              their 'config' property.";
         }
         choice schema-ref {
           mandatory true;
           description
             "Alternatives for specifying the schema.";
           leaf inline {
             type empty;
             description
               "This leaf indicates that the server has mounted
                'ietf-yang-library' and 'ietf-schema-mount' at the mount
                point, and their instantiation (i.e., state data
                containers 'yanglib:modules-state' and 'schema-mounts')
                provides the information about the mounted schema.";
           }
           list use-schema {
             key "name";
             min-elements 1;
             description
               "Each entry of this list contains a reference to a schema
                defined in the /schema-mounts/schema list.";
             leaf name {
               type leafref {
                 path "/schema-mounts/schema/name";
               }
               description
                 "Name of the referenced schema.";
             }
             leaf-list parent-reference {
               type yang:xpath1.0;
               description
                 "Entries of this leaf-list are XPath 1.0 expressions
                  that are evaluated in the following context:

                  - The context node is the node in the parent data tree
                    where the mount-point is defined.

                  - The accessible tree is the parent data tree
                    *without* any nodes defined in modules that are
                    mounted inside the parent schema.

                  - The context position and context size are both equal
                    to 1.
```

```
                  - The set of variable bindings is empty.

                  - The function library is the core function library
                    defined in [XPath] and the functions defined in
                    Section 10 of [RFC7950].

                  - The set of namespace declarations is defined by the
                    'namespace' list under 'schema-mounts'.

                  Each XPath expression MUST evaluate to a nodeset
                  (possibly empty). For the purposes of evaluating XPath
                  expressions whose context nodes are defined in the
                  mounted schema, the union of all these nodesets
                  together with ancestor nodes are added to the
                  accessible data tree.";
             }
           }
         }
       }
     }

     /*
      * State data nodes
      */

     container schema-mounts {
       config false;
       description
         "Contains information about the structure of the overall
          mounted data model implemented in the server.";
       list namespace {
         key "prefix";
         description
           "This list provides a mapping of namespace prefixes that are
            used in XPath expressions of 'parent-reference' leafs to the
            corresponding namespace URI references.";
         leaf prefix {
           type yang:yang-identifier;
           description
             "Namespace prefix.";
         }
         leaf uri {
           type inet:uri;
           description
             "Namespace URI reference.";
         }
       }
       uses mount-point-list;
```

```
       list schema {
         key "name";
         description
           "Each entry specifies a schema that can be mounted at a mount
            point.  The schema information consists of two parts:

            - an instance of YANG library that defines YANG modules used
              in the schema,

            - mount-point list with content identical to the top-level
              mount-point list (this makes the schema structure
              recursive).";
         leaf name {
           type string;
           description
             "Arbitrary name of the schema entry.";
         }
         uses yanglib:module-list;
         uses mount-point-list;
       }
     }
   }

   <CODE ENDS>
```

9. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount

Registrant Contact: The IESG.

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

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

```
     name:         ietf-yang-schema-mount
     namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount
     prefix:       yangmnt
     reference:    RFC XXXX
```

10. Security Considerations

This document defines a mechanism for combining schemas from different YANG modules into a single schema, and as such doesn't introduce new security issues.

11. Contributors

The idea of having some way to combine schemas from different YANG modules into one has been proposed independently by several groups of people: Alexander Clemm, Jan Medved, and Eric Voit ([I-D.clemm-netmod-mount]); and Lou Berger and Christian Hopps:

- Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- Alexander Clemm, Huawei, <alexander.clemm@huawei.com>
- Christian Hopps, Deutsche Telekom, <chopps@chopps.org>
- Jan Medved, Cisco, <jmedved@cisco.com>
- Eric Voit, Cisco, <evoit@cisco.com>

12. References

12.1. Normative References

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

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

[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>.

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

[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>.

[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>.

12.2. Informative References

[I-D.clemm-netmod-mount]
Clemm, A., Voit, E., and J. Medved, "Mounting YANG-Defined Information from Remote Datastores", draft-clemm-netmod-mount-06 (work in progress), March 2017.

[I-D.ietf-isis-yang-isis-cfg]
Litkowski, S., Yeung, D., Lindem, A., Zhang, Z., and L. Lhotka, "YANG Data Model for IS-IS protocol", draft-ietf-isis-yang-isis-cfg-17 (work in progress), March 2017.

[I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-01 (work in progress), June 2017.

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.

[I-D.ietf-rtgwg-lne-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Logical Network Elements", draft-ietf-rtgwg-lne-model-03 (work in progress), July 2017.

[I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Network Instances", draft-ietf-rtgwg-ni-model-03 (work in progress), July 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>.

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

[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>.

Appendix A. Example: Device Model with LNEs and NIs

This non-normative example demonstrates an implementation of the device model as specified in Section 2 of [I-D.ietf-rtgwg-device-model], using both logical network elements (LNE) and network instances (NI).

A.1. Physical Device

The data model for the physical device may be described by this YANG library content:


```
"ietf-yang-library:modules-state": {
  "module-set-id": "14e2ab5dc325f6d86f743e8d3ade233f1a61a899",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
      "revision": "2014-05-08",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ip",
      "revision": "2014-06-16",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-logical-network-element",
      "revision": "2016-10-21",
      "feature": [
        "bind-lne-name"
```

```
         ],
         "namespace":
           "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
         "conformance-type": "implement"
       },
       {
         "name": "ietf-yang-library",
         "revision": "2016-06-21",
         "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
         "conformance-type": "implement"
       },
       {
         "name": "ietf-yang-schema-mount",
         "revision": "2017-05-16",
         "namespace":
           "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
         "conformance-type": "implement"
       },
       {
         "name": "ietf-yang-types",
         "revision": "2013-07-15",
         "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
         "conformance-type": "import"
       }
     ]
   }
```

A.2. Logical Network Elements

Each LNE can have a specific data model that is determined at run time, so it is appropriate to mount it using the "inline" method, hence the following "schema-mounts" data:

```
   "ietf-yang-schema-mount:schema-mounts": {
     "mount-point": [
       {
         "module": "ietf-logical-network-element",
         "label": "root",
         "inline": [null]
       }
     ]
   }
```

An administrator of the host device has to configure an entry for each LNE instance, for example,

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-logical-network-element:bind-lne-name": "eth0"
      }
    ]
  },
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "lne-1",
        "managed": true,
        "description": "LNE with NIs",
        "root": {
          ...
        }
      },
      ...
    ]
  }
}
```


and then also place necessary state data as the contents of the "root" instance, which should include at least

o YANG library data specifying the LNE's data model, for example:


```
"ietf-yang-library:modules-state": {
  "module-set-id": "9358e11874068c8be06562089e94a89e0a392019",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-inet-types",
      "revision": "2013-07-15",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
      "conformance-type": "import"
    },
    {
      "name": "ietf-interfaces",
```

```
          "revision": "2014-05-08",
          "feature": [
            "arbitrary-names",
            "pre-provisioning"
          ],
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ip",
          "revision": "2014-06-16",
          "feature": [
            "ipv6-privacy-autoconf"
          ],
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-network-instance",
          "revision": "2016-10-27",
          "feature": [
            "bind-network-instance-name"
          ],
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-network-instance",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-library",
          "revision": "2016-06-21",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-schema-mount",
          "revision": "2017-05-16",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-yang-types",
          "revision": "2013-07-15",
          "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
          "conformance-type": "import"
        }
      ]
    }
```

o state data for interfaces assigned to the LNE instance (that effectively become system-controlled interfaces for the LNE), for example:


```
"ietf-interfaces:interfaces-state": {
  "interface": [
    {
      "name": "eth0",
      "type": "iana-if-type:ethernetCsmacd",
      "oper-status": "up",
      "statistics": {
        "discontinuity-time": "2016-12-16T17:11:27+02:00"
      },
      "ietf-ip:ipv6": {
        "address": [
          {
            "ip": "fe80::42a8:f0ff:fea8:24fe",
            "origin": "link-layer",
            "prefix-length": 64
          }
        ]
      }
    },
    ...
  ]
}
```


A.3. Network Instances

Assuming that network instances share the same data model, it can be mounted using the "use-schema" method as follows:


```
"ietf-yang-schema-mount:schema-mounts": {
  "namespace": [
    {
        "prefix": "if",
        "uri": "urn:ietf:params:xml:ns:yang:ietf-interfaces"
    },
    {
        "prefix": "ni",
        "uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
    }
  ],
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "root",
      "use-schema": [
```

```
          {
            "name": "ni-schema",
            "parent-reference": [
              "/if:interfaces/if:interface[
                 ni:bind-network-instance-name = current()/../ni:name]"
            ]
          }
        ]
      }
    ],
    "schema": [
      {
        "name": "ni-schema",
        "module": [
          {
            "name": "ietf-ipv4-unicast-routing",
            "revision": "2016-11-04",
            "namespace":
             "urn:ietf:params:xml:ns:yang:ietf-ipv4-unicast-routing",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-ipv6-unicast-routing",
            "revision": "2016-11-04",
            "namespace":
             "urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-routing",
            "revision": "2016-11-04",
            "feature": [
              "multiple-ribs",
              "router-id"
            ],
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-routing",
            "conformance-type": "implement"
          }
        ]
      }
    ]
  }
```


Note also that the "ietf-interfaces" module appears in the "parent-reference" leaf-list for the mounted NI schema. This means that references to LNE interfaces, such as "outgoing-interface" in static routes, are valid despite the fact that "ietf-interfaces" isn’t part of the NI schema.

A.4. Invoking an RPC Operation

Assume that the mounted NI data model also implements the "ietf-isis" module [I-D.ietf-isis-yang-isis-cfg]. An RPC operation defined in this module, such as "clear-adjacency", can be invoked by a client session of a LNE's RESTCONF server as an action tied to a the mount point of a particular network instance using a request URI like this (all on one line):

```
POST /restconf/data/ietf-network-instance:network-instances/
    network-instance=rtrA/root/ietf-isis:clear-adjacency HTTP/1.1
```

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 20, 2019

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC

October 17, 2018

YANG Schema Mount
draft-ietf-netmod-schema-mount-12

Abstract

This document defines a mechanism to add the schema trees defined by a set of YANG modules onto a mount point defined in the schema tree in some YANG module.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Modularity and extensibility were among the leading design principles of the YANG data modeling language. As a result, the same YANG module can be combined with various sets of other modules and thus form a data model that is tailored to meet the requirements of a specific use case. Server implementors are only required to specify all YANG modules comprising the data model (together with their revisions and other optional choices) in the YANG library data ([RFC7895], [I-D.ietf-netconf-rfc7895bis] and Section 5.6.4 of [RFC7950]) implemented by the server. Such YANG modules appear in the data model "side by side", i.e., top-level data nodes of each module - if there are any - are also top-level nodes of the overall data model.

YANG has two mechanisms for contributing a schema hierarchy defined elsewhere to the contents of an internal node of the schema tree; these mechanisms are realized through the following YANG statements:

- o The "uses" statement explicitly incorporates the contents of a grouping defined in the same or another module. See Section 4.2.6 of [RFC7950] for more details.

- o The "augment" statement explicitly adds contents to a target node defined in the same or another module. See Section 4.2.8 of [RFC7950] for more details.

With both mechanisms, the YANG module with the "uses" or "augment" statement explicitly defines the exact location in the schema tree where the new nodes are placed.

In some cases these mechanisms are not sufficient; it is sometimes necessary that an existing module (or a set of modules) is added to the data model starting at locations other than the root. For example, YANG modules such as "ietf-interfaces" [RFC8343] are defined so as to be used in a data model of a physical device. Now suppose we want to model a device that supports multiple logical devices [I-D.ietf-rtgwg-lne-model], each of which has its own instantiation of "ietf-interfaces", and possibly other modules, but, at the same time, we want to be able to manage all these logical devices from the master device. Hence, we would like to have a schema tree like this:

```
+--rw interfaces
|  +--rw interface* [name]
|     ...
+--rw logical-network-element* [name]
   +--rw name
   |   ...
   +--rw interfaces
     +--rw interface* [name]
         ...
```

With the "uses" approach, the complete schema tree of "ietf-interfaces" would have to be wrapped in a grouping, and then this grouping would have to be used at the top level (for the master device) and then also in the "logical-network-element" list (for the logical devices). This approach has several disadvantages:

- o It is not scalable because every time there is a new YANG module that needs to be added to the logical device model, we have to update the model for logical devices with another "uses" statement pulling in contents of the new module.

o Absolute references to nodes defined inside a grouping may break if the grouping is used in different locations.

o Nodes defined inside a grouping belong to the namespace of the module where it is used, which makes references to such nodes from other modules difficult or even impossible.

o It would be difficult for vendors to add proprietary modules when the "uses" statements are defined in a standard module.

With the "augment" approach, "ietf-interfaces" would have to augment the "logical-network-element" list with all its nodes, and at the same time define all its nodes at the top level. The same hierarchy of nodes would thus have to be defined twice, which is clearly not scalable either.

This document introduces a new mechanism, denoted as schema mount, that allows for mounting one data model consisting of any number of YANG modules at a specified location of another (parent) schema. Unlike the "uses" and "augment" approaches discussed above, the mounted modules needn’t be specially prepared for mounting and, consequently, existing modules such as "ietf-interfaces" can be mounted without any modifications.

The basic idea of schema mount is to label a data node in the parent schema as the mount point, and then define a complete data model to be attached to the mount point so that the labeled data node effectively becomes the root node of the mounted data model.

In principle, the mounted schema can be specified at three different phases of the data model life cycle:

1. Design-time: the mounted schema is defined along with the mount point in the parent YANG module. In this case, the mounted schema has to be the same for every implementation of the parent module.

2. Implementation-time: the mounted schema is defined by a server implementor and is as stable as the YANG library information of the server.

3. Run-time: the mounted schema is defined by instance data that is part of the mounted data model. If there are multiple instances of the same mount point (e.g., in multiple entries of a list), the mounted data model may be different for each instance.

The schema mount mechanism defined in this document provides support only for the latter two cases. Design-time mounts are outside the scope of this document, and could be possibly dealt with in a future revision of the YANG data modeling language.

Schema mount applies to the data model, and specifically does not assume anything about the source of instance data for the mounted schemas. It may be implemented using the same instrumentation as the rest of the system, or it may be implemented by querying some other system. Future specifications may define mechanisms to control or monitor the implementation of specific mount points.

How and when specific mount points are instantiated by the server is out of scope for this document. Such mechanisms may be defined in future specifications.

This document allows mounting of complete data models only. Other specifications may extend this model by defining additional mechanisms such as mounting sub-hierarchies of a module.

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

2. Terminology and Notation

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

The following terms are defined in [RFC7950] and are not redefined here:

- action
- container
- data node
- list
- RPC operation
- schema node
- schema tree

The following terms are defined in [RFC8342] and are not redefined here:

- o client

- o notification

- o operational state

- o server

The following term is defined in [RFC8343] and is not redefined here:

- o system-controlled interface

The following term is defined in [I-D.ietf-netconf-rfc7895bis] is not redefined here:

- o YANG library content identifier

The following additional terms are used within this document:

- o mount point: A container or a list node whose definition contains the "mount-point" extension statement. The argument of the "mount-point" statement defines a label for the mount point.

- o schema: A collection of schema trees with a common root.

- o top-level schema: A schema rooted at the root node.

- o mounted schema: A schema rooted at a mount point.

- o parent schema (of a mounted schema): A schema containing the mount point.

- o schema mount: The mechanism to combine data models defined in this document.

2.1. Tree Diagrams

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

2.2. Namespace Prefixes

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

Prefix	YANG module	Reference
yangmnt	ietf-yang-schema-mount	Section 9
inet	ietf-inet-types	[RFC6991]
yang	ietf-yang-types	[RFC6991]
yanglib	ietf-yang-library	[RFC7895], [I-D.ietf-netconf-rfc7895bis]

Table 1: Namespace Prefixes

3. Schema Mount

The schema mount mechanism defined in this document provides a new extensibility mechanism for use with YANG 1.1. In contrast to the existing mechanisms described in Section 1, schema mount defines the relationship between the source and target YANG modules outside these modules. The procedure consists of two separate steps that are described in the following subsections.

3.1. Mount Point Definition

A "container" or "list" node becomes a mount point if the "mount-point" extension (defined in the "ietf-yang-schema-mount" module) is used in its definition. This extension can appear only as a substatement of "container" and "list" statements.

The argument of the "mount-point" extension is a YANG identifier that defines a label for the mount point. A module MAY contain multiple "mount-point" statements having the same argument.

It is therefore up to the designer of the parent schema to decide about the placement of mount points. A mount point can also be made conditional by placing "if-feature" and/or "when" as substatements of the "container" or "list" statement that represents the mount point.

The "mount-point" statement MUST NOT be used in a YANG version 1 module [RFC6020]. The reason for this is that otherwise it is not possible to invoke mounted RPC operations, and receive mounted notifications. See Section 5 for details. Note, however, that modules written in any YANG version, including version 1, can be mounted under a mount point.

Note that the "mount-point" statement does not define a new data node.

3.2. Data Model

This document defines the YANG 1.1 module [RFC7950] "ietf-yang-schema-mount", which has the following structure:

```
module: ietf-yang-schema-mount
  +--ro schema-mounts
     +--ro namespace* [prefix]
     |  +--ro prefix    yang:yang-identifier
     |  +--ro uri?      inet:uri
     +--ro mount-point* [module label]
        +--ro module                 yang:yang-identifier
        +--ro label                  yang:yang-identifier
        +--ro config?                boolean
        +--ro (schema-ref)
           +--:(inline)
           |  +--ro inline!
           +--:(shared-schema)
              +--ro shared-schema!
                 +--ro parent-reference*   yang:xpath1.0
```

3.3. Specification of the Mounted Schema

Mounted schemas for all mount points in the parent schema are determined from state data in the "/schema-mounts" container.

Generally, the modules that are mounted under a mount point have no relation to the modules in the parent schema; specifically, if a module is mounted it may or may not be present in the parent schema and, if present, its data will generally have no relationship to the data of the parent. Exceptions are possible and such needs to be defined in the model defining the exception. For example, [I-D.ietf-rtgwg-lne-model] defines a mechanism to bind interfaces to mounted logical network elements.

The "/schema-mounts" container has the "mount-point" list as one of its children. Every entry of this list refers through its key to a mount point and specifies the mounted schema.

If a mount point is defined in the parent schema but does not have an entry in the "mount-point" list, then the mounted schema is void, i.e., instances of that mount point MUST NOT contain any data except those that are defined in the parent schema.

If multiple mount points with the same name are defined in the same module - either directly or because the mount point is defined in a grouping and the grouping is used multiple times - then the corresponding "mount-point" entry applies equally to all such mount points.

The "config" property of mounted schema nodes is overridden and all nodes in the mounted schema are read-only ("config false") if at least one of the following conditions is satisfied for a mount point:

- o the mount point is itself defined as "config false"
- o the "config" leaf in the corresponding entry of the "mount-point" list is set to "false".

An entry of the "mount-point" list can specify the mounted schema in two different ways, "inline" or "shared-schema".

The mounted schema is determined at run time: every instance of the mount point that exists in the operational state MUST contain a copy of YANG library data that defines the mounted schema exactly as for a top-level schema. A client is expected to retrieve this data from the instance tree. In the "inline" case, instances of the same mount point MAY use different mounted schemas, whereas in the "shared-schema" case, all instances MUST use the same mounted schema. This means that in the "shared-schema" case, all instances of the same mount point MUST have the same YANG library content identifier. In the "inline" case, if two instances have the same YANG library content identifier it is not guaranteed that the YANG library contents are equal for these instances.

Examples of "inline" and "shared-schema" can be found in Appendix A.2 and Appendix A.3, respectively.

3.4. Multiple Levels of Schema Mount

YANG modules in a mounted schema MAY again contain mount points under which other schemas can be mounted. Consequently, it is possible to construct data models with an arbitrary number of mounted schemas. A schema for a mount point contained in a mounted module can be specified by implementing "ietf-yang-library" and "ietf-yang-schema-mount" modules in the mounted schema, and specifying the schemas exactly as it is done in the top-level schema.

4. Referring to Data Nodes in the Parent Schema

A fundamental design principle of schema mount is that the mounted schema works exactly as a top-level schema, i.e., it is confined to the "mount jail". This means that all paths in the mounted schema (in leafrefs, instance-identifiers, XPath [XPATH] expressions, and target nodes of augments) are interpreted with the mount point as the root node. YANG modules of the mounted schema as well as corresponding instance data thus cannot refer to schema nodes or instance data outside the mount jail.

However, this restriction is sometimes too severe. A typical example is network instances (NI) [I-D.ietf-rtgwg-ni-model], where each NI has its own routing engine but the list of interfaces is global and shared by all NIs. If we want to model this organization with the NI schema mounted using schema mount, the overall schema tree would look schematically as follows:

```
+--rw interfaces
|  +--rw interface* [name]
|     ...
+--rw network-instances
   +--rw network-instance* [name]
      +--rw name
      +--rw root
         +--rw routing
            ...
```

Here, the "root" node is the mount point for the NI schema. Routing configuration inside an NI often needs to refer to interfaces (at least those that are assigned to the NI), which is impossible unless such a reference can point to a node in the parent schema (interface name).

Therefore, schema mount also allows for such references. For every mount point in the "shared-schema" case, it is possible to specify a leaf-list named "parent-reference" that contains zero or more XPath 1.0 expressions. Each expression is evaluated with the node in the parent data tree where the mount point is defined as the context node. The result of this evaluation MUST be a nodeset (see the description of the "parent-reference" node for a complete definition of the evaluation context). For the purposes of evaluating XPath expressions within the mounted data tree, the union of all such nodesets is added to the accessible data tree.

It is worth emphasizing that the nodes specified in "parent-reference" leaf-list are available in the mounted schema only for XPath evaluations. In particular, they cannot be accessed there via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040].

5. RPC operations and Notifications

If a mounted YANG module defines an RPC operation, clients can invoke this operation as if it were defined as an action for the corresponding mount point, see Section 7.15 of [RFC7950]. An example of this is given in Appendix A.4.

Similarly, if the server emits a notification defined at the top level of any mounted module, it MUST be represented as if the notification was connected to the mount point, see Section 7.16 of [RFC7950].

Note, inline actions and notifications will not work when they are contained within a list node without a "key" statement (see section 7.15 and 7.16 of [RFC7950]). Therefore, to be useful, mount points that contain modules with RPCs, actions, and notifications SHOULD NOT have any ancestor node that is a list node without a "key" statement. This requirement applies to the definition of modules using the "mount-point" extension statement.

6. Network Management Datastore Architecture (NMDA) Considerations

The schema mount solution presented in this document is designed to work both with servers that implement the NMDA [RFC8342], and old servers that don't implement the NMDA.

Note to RFC Editor: please update the date YYYY-MM-DD below with the revision of the ietf-yang-library in the published version of draft-ietf-netconf-rfc7895bis, and remove this note.

Specifically, a server that doesn't support the NMDA, MAY implement revision 2016-06-21 of "ietf-yang-library" [RFC7895] under a mount point. A server that supports the NMDA, MUST implement at least revision YYYY-MM-DD of "ietf-yang-library" [I-D.ietf-netconf-rfc7895bis] under the mount points.

7. Interaction with the Network Configuration Access Control Model (NACM)

If NACM [RFC8341] is implemented on a server, it is used to control access to nodes defined by the mounted schema in the same way as for nodes defined by the top-level schema.

For example, suppose the module "ietf-interfaces" is mounted in the "root" container in the "logical-network-element" list defined in [I-D.ietf-rtgwg-lne-model]. Then the following NACM path can be used to control access to the "interfaces" container (where the character

'\' is used where a line break has been inserted for formatting reasons):

```
<path xmlns:lne=
        "urn:ietf:params:xml:ns:yang:ietf-logical-network-element"
      xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
  /lne:logical-network-elements\
    /lne:logical-network-element/lne:root/if:interfaces
</path>
```

8. Implementation Notes

Network management of devices that use a data model with schema mount can be implemented in different ways. However, the following implementations options are envisioned as typical:

- o shared management: instance data of both parent and mounted schemas are accessible within the same management session.

- o split management: one (master) management session has access to instance data of both parent and mounted schemas but, in addition, an extra session exists for every instance of the mount point, having access only to the mounted data tree.

9. Schema Mount YANG Module

This module references [RFC6991].

<CODE BEGINS> file "ietf-yang-schema-mount@2018-10-16"

```
module ietf-yang-schema-mount {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount";
  prefix yangmnt;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  organization
```

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

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

     Editor:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

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

  // RFC Ed.: replace XXXX with actual RFC number and
  // remove this note.
  description
    "This module defines a YANG extension statement that can be used
     to incorporate data models defined in other YANG modules in a
     module. It also defines operational state data that specify the
     overall structure of the data model.

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

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
     'MAY', and 'OPTIONAL' in the module text are to be interpreted
     as described in BCP 14 [RFC 2119] [RFC8174] when, and only when,
     they appear in all capitals, as shown here.

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

  // RFC Ed.: update the date below with the date of RFC publication
  // and remove this note.
  revision 2018-10-16 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: YANG Schema Mount";
  }
```

```
     /*
      * Extensions
      */

     extension mount-point {
       argument label;
       description
         "The argument 'label' is a YANG identifier, i.e., it is of the
          type 'yang:yang-identifier'.

          The 'mount-point' statement MUST NOT be used in a YANG
          version 1 module, neither explicitly nor via a 'uses'
          statement.

          The 'mount-point' statement MAY be present as a substatement
          of 'container' and 'list', and MUST NOT be present elsewhere.
          There MUST NOT be more than one 'mount-point' statement in a
          given 'container' or 'list' statement.

          If a mount point is defined within a grouping, its label is
          bound to the module where the grouping is used.

          A mount point defines a place in the node hierarchy where
          other data models may be attached. A server that implements a
          module with a mount point populates the
          /schema-mounts/mount-point list with detailed information on
          which data models are mounted at each mount point.

          Note that the 'mount-point' statement does not define a new
          data node.";
     }

     /*
      * State data nodes
      */

     container schema-mounts {
       config false;
       description
         "Contains information about the structure of the overall
          mounted data model implemented in the server.";
       list namespace {
         key "prefix";
         description
           "This list provides a mapping of namespace prefixes that are
            used in XPath expressions of 'parent-reference' leafs to the
            corresponding namespace URI references.";
         leaf prefix {
```

```
          type yang:yang-identifier;
          description
            "Namespace prefix.";
        }
        leaf uri {
          type inet:uri;
          description
            "Namespace URI reference.";
        }
      }
      list mount-point {
        key "module label";
        description
          "Each entry of this list specifies a schema for a particular
           mount point.

           Each mount point MUST be defined using the 'mount-point'
           extension in one of the modules listed in the server's
           YANG library instance with conformance type 'implement'.";
        leaf module {
          type yang:yang-identifier;
          description
            "Name of a module containing the mount point.";
        }
        leaf label {
          type yang:yang-identifier;
          description
            "Label of the mount point defined using the 'mount-point'
             extension.";
        }
        leaf config {
          type boolean;
          default "true";
          description
            "If this leaf is set to 'false', then all data nodes in the
             mounted schema are read-only (config false), regardless of
             their 'config' property.";
        }
        choice schema-ref {
          mandatory true;
          description
            "Alternatives for specifying the schema.";
          container inline {
            presence
              "A complete self-contained schema is mounted at the
               mount point.";
            description
              "This node indicates that the server has mounted at least
```

```
             the module 'ietf-yang-library' at the mount point, and
             its instantiation provides the information about the
             mounted schema.

             Different instances of the mount point may have
             different schemas mounted.";
        }
        container shared-schema {
          presence
            "The mounted schema together with the 'parent-reference'
             make up the schema for this mount point.";
          description
            "This node indicates that the server has mounted at least
             the module 'ietf-yang-library' at the mount point, and
             its instantiation provides the information about the
             mounted schema.  When XPath expressions in the mounted
             schema are evaluated, the 'parent-reference' leaf-list
             is taken into account.

             Different instances of the mount point MUST have the
             same schema mounted.";
          leaf-list parent-reference {
            type yang:xpath1.0;
            description
              "Entries of this leaf-list are XPath 1.0 expressions
               that are evaluated in the following context:

               - The context node is the node in the parent data tree
                 where the mount-point is defined.

               - The accessible tree is the parent data tree
                 *without* any nodes defined in modules that are
                 mounted inside the parent schema.

               - The context position and context size are both equal
                 to 1.

               - The set of variable bindings is empty.

               - The function library is the core function library
                 defined in [XPath] and the functions defined in
                 Section 10 of [RFC7950].

               - The set of namespace declarations is defined by the
                 'namespace' list under 'schema-mounts'.

               Each XPath expression MUST evaluate to a nodeset
               (possibly empty). For the purposes of evaluating XPath
```

```
                  expressions whose context nodes are defined in the
                  mounted schema, the union of all these nodesets
                  together with ancestor nodes are added to the
                  accessible data tree.

                  Note that in the case 'ietf-yang-schema-mount' is
                  itself mounted, a 'parent-reference' in the mounted
                  module may refer to nodes that were brought into the
                  accessible tree through a 'parent-reference' in the
                  parent schema.";
             }
           }
         }
       }
     }
   }

   <CODE ENDS>
```

10. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount

Registrant Contact: The IESG.

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

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

```
  name:         ietf-yang-schema-mount
  namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount
  prefix:       yangmnt
  reference:    RFC XXXX
```

11. Security Considerations

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

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

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

o /schema-mounts: The schema defined by this state data provides detailed information about a server implementation may help an attacker identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules included in the schema, module revisions, module features, or even module deviations. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the schema information will help an attacker identify server implementations with such a defect, in order to launch a denial-of-service attack on the device.

It is important to take the security considerations for all nodes in the mounted schemas into account, and control access to these nodes by using the mechanism described in Section 7.

Care must be taken when the "parent-reference" XPath expressions are constructed, since the result of the evaluation of these expressions is added to the accessible tree for any XPath expression found in the mounted schema.

12. Contributors

The idea of having some way to combine schemas from different YANG modules into one has been proposed independently by several groups of people: Alexander Clemm, Jan Medved, and Eric Voit ([I-D.clemm-netmod-mount]); and Lou Berger and Christian Hopps:

o Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>

o Alexander Clemm, Huawei, <alexander.clemm@huawei.com>

o Christian Hopps, Deutsche Telekom, <chopps@chopps.org>

o Jan Medved, Cisco, <jmedved@cisco.com>

o Eric Voit, Cisco, <evoit@cisco.com>

13. References

13.1. Normative References

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-06 (work in progress), April 2018.

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

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

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

[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>.

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

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

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

[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>.

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

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

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

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

13.2. Informative References

[I-D.clemm-netmod-mount]
Clemm, A., Voit, E., and J. Medved, "Mounting YANG-Defined Information from Remote Datastores", draft-clemm-netmod-mount-06 (work in progress), March 2017.

[I-D.ietf-isis-yang-isis-cfg]
Litkowski, S., Yeung, D., Lindem, A., Zhang, Z., and L. Lhotka, "YANG Data Model for IS-IS protocol", draft-ietf-isis-yang-isis-cfg-24 (work in progress), August 2018.

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.

[I-D.ietf-rtgwg-lne-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Logical Network Elements", draft-ietf-rtgwg-lne-model-10 (work in progress), March 2018.

[I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Network Instances", draft-ietf-rtgwg-ni-model-12 (work in progress), March 2018.

[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>.

[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>.

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

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

Appendix A. Example: Device Model with LNEs and NIs

This non-normative example demonstrates an implementation of the device model as specified in Section 2 of [I-D.ietf-rtgwg-device-model], using both logical network elements (LNE) and network instances (NI).

In these examples, the character '\' is used where a line break has been inserted for formatting reasons.

A.1. Physical Device

The data model for the physical device may be described by this YANG library content, assuming the server supports the NMDA:


```
{
  "ietf-yang-library:yang-library": {
    "content-id": "14e2ab5dc325f6d86f743e8d3ade233f1a61a899",
    "module-set": [
      {
        "name": "physical-device-modules",
        "module": [
          {
            "name": "ietf-datastores",
            "revision": "2018-02-14",
            "namespace":
              "urn:ietf:params:xml:ns:yang:ietf-datastores"
          },
          {
            "name": "iana-if-type",
            "revision": "2015-06-12",
            "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type"
          },
          {
            "name": "ietf-interfaces",
            "revision": "2018-02-20",
```

```
                "feature": ["arbitrary-names", "pre-provisioning" ],
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-interfaces"
              },
              {
                "name": "ietf-ip",
                "revision": "2018-02-22",
                "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip"
              },
              {
                "name": "ietf-logical-network-element",
                "revision": "2016-10-21",
                "feature": [ "bind-lne-name" ],
                "namespace":
                  "urn:ietf:params:xml:ns:yang:\
                  ietf-logical-network-element"
              },
              {
                "name": "ietf-yang-library",
                "revision": "2018-02-21",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-library"
              },
              {
                "name": "ietf-yang-schema-mount",
                "revision": "2018-03-20",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount"
              }
            ],
            "import-only-module": [
              {
                "name": "ietf-inet-types",
                "revision": "2013-07-15",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-inet-types"
              },
              {
                "name": "ietf-yang-types",
                "revision": "2013-07-15",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-types"
              }
            ]
          }
        ],
        "schema": [
          {
```

```
            "name": "physical-device-schema",
            "module-set": [ "physical-device-modules" ]
          }
        ],
        "datastore": [
          {
            "name": "ietf-datastores:running",
            "schema": "physical-device-schema"
          },
          {
            "name": "ietf-datastores:operational",
            "schema": "physical-device-schema"
          }
        ]
      }
   }
```


A.2. Logical Network Elements

Each LNE can have a specific data model that is determined at run time, so it is appropriate to mount it using the "inline" method, hence the following "schema-mounts" data:


```
   {
     "ietf-yang-schema-mount:schema-mounts": {
       "mount-point": [
         {
           "module": "ietf-logical-network-element",
           "label": "root",
           "inline": {}
         }
       ]
     }
   }
```


An administrator of the host device has to configure an entry for each LNE instance, for example,

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-logical-network-element:bind-lne-name": "eth0"
      }
    ]
  },
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "lne-1",
        "managed": true,
        "description": "LNE with NIs",
        "root": {
          ...
        }
      }
      ...
    ]
  }
}
```


and then also place necessary state data as the contents of the "root" instance, which should include at least

o YANG library data specifying the LNE’s data model, for example, assuming the server does not implement the NMDA:


```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "9358e11874068c8be06562089e94a89e0a392019",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace": "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-inet-types",
        "revision": "2013-07-15",
        "namespace": "urn:ietf:params:xml:ns:yang:ietf-inet-types",
        "conformance-type": "import"
      },
```

```
         {
           "name": "ietf-interfaces",
           "revision": "2014-05-08",
           "feature": [
             "arbitrary-names",
             "pre-provisioning"
           ],
           "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",
           "conformance-type": "implement"
         },
         {
           "name": "ietf-ip",
           "revision": "2014-06-16",
           "feature": [
             "ipv6-privacy-autoconf"
           ],
           "namespace": "urn:ietf:params:xml:ns:yang:ietf-ip",
           "conformance-type": "implement"
         },
         {
           "name": "ietf-network-instance",
           "revision": "2016-10-27",
           "feature": [
             "bind-network-instance-name"
           ],
           "namespace":
             "urn:ietf:params:xml:ns:yang:ietf-network-instance",
           "conformance-type": "implement"
         },
         {
           "name": "ietf-yang-library",
           "revision": "2016-06-21",
           "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-library",
           "conformance-type": "implement"
         },
         {
           "name": "ietf-yang-schema-mount",
           "revision": "2017-05-16",
           "namespace":
             "urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount",
           "conformance-type": "implement"
         },
         {
           "name": "ietf-yang-types",
           "revision": "2013-07-15",
           "namespace": "urn:ietf:params:xml:ns:yang:ietf-yang-types",
           "conformance-type": "import"
         }
```

```
      ]
    }
  }
```


o state data for interfaces assigned to the LNE instance (that effectively become system-controlled interfaces for the LNE), for example:


```
  {
    "ietf-interfaces:interfaces": {
      "interface": [
        {
          "name": "eth0",
          "type": "iana-if-type:ethernetCsmacd",
          "oper-status": "up",
          "statistics": {
            "discontinuity-time": "2016-12-16T17:11:27+02:00"
          },
          "ietf-ip:ipv6": {
            "address": [
              {
                "ip": "fe80::42a8:f0ff:fea8:24fe",
                "origin": "link-layer",
                "prefix-length": 64
              }
            ]
          }
        }
      ]
    }
  }
```


A.3. Network Instances

Assuming that network instances share the same data model, it can be mounted using the "shared-schema" method as follows:

```
{
  "ietf-yang-schema-mount:schema-mounts": {
    "namespace": [
      {
          "prefix": "if",
          "uri": "urn:ietf:params:xml:ns:yang:ietf-interfaces"
      },
      {
          "prefix": "ni",
          "uri": "urn:ietf:params:xml:ns:yang:ietf-network-instance"
      }
    ],
    "mount-point": [
      {
        "module": "ietf-network-instance",
        "label": "root",
          "shared-schema": {
            "parent-reference": [
              "/if:interfaces/if:interface[\
              ni:bind-network-instance-name = current()/../ni:name]"
            ]
          }
      }
    ]
  }
}
```


Note also that the "ietf-interfaces" module appears in the "parent-reference" leaf-list for the mounted NI schema. This means that references to LNE interfaces, such as "outgoing-interface" in static routes, are valid despite the fact that "ietf-interfaces" isn’t part of the NI schema.

A.4. Invoking an RPC Operation

Assume that the mounted NI data model also implements the "ietf-isis" module [I-D.ietf-isis-yang-isis-cfg]. An RPC operation defined in this module, such as "clear-adjacency", can be invoked by a client session of a LNE’s RESTCONF server as an action tied to a the mount point of a particular network instance using a request URI like this (all on one line):

```
POST /restconf/data/ietf-network-instance:network-instances/
    network-instance=rtrA/root/ietf-isis:clear-adjacency HTTP/1.1
```

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

Network Working Group   M. Bjorklund
Internet-Draft   Tail-f Systems
Intended status: Standards Track   L. Berger, Ed.
Expires: April 28, 2018   LabN Consulting, L.L.C.
                                                        October 25, 2017

YANG Tree Diagrams
draft-ietf-netmod-yang-tree-diagrams-02

Abstract

This document captures the current syntax used in YANG module Tree Diagrams. The purpose of the document is to provide a single location for this definition. This syntax may be updated from time to time based on the evolution of the YANG language.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

YANG Tree Diagrams were first published in [RFC7223]. Such diagrams are commonly used to provided a simplified graphical representation of a data model and can be automatically generated via tools such as "pyang". (See <https://github.com/mbj4668/pyang>). This document provides the syntax used in YANG Tree Diagrams. It is expected that this document will be updated or replaced as changes to the YANG language, see [RFC7950], necessitate.

Today's common practice is include the definition of the syntax used to represent a YANG module in every document that provides a tree diagram. This practice has several disadvantages and the purpose of the document is to provide a single location for this definition. It is not the intent of this document to restrict future changes, but rather to ensure such changes are easily identified and suitably agreed upon.

An example tree diagram can be found in [RFC7223] Section 3. A portion of which follows:

```
+--rw interfaces
|  +--rw interface* [name]
|     +--rw name                        string
|     +--rw description?                string
|     +--rw type                        identityref
|     +--rw enabled?                    boolean
|     +--rw link-up-down-trap-enable?   enumeration
```

2. Tree Diagram Syntax

This section provides the meaning of the symbols used in YANG Tree diagrams.

A full tree diagram of a module represents all elements. It includes the name of the module and sections for top level module statements (typically containers), augmentations, rpcs and notifications all identified under a module statement. Module trees may be included in a document as a whole, by one or more sections, or even subsets of nodes.

A module is identified by "module:" followed the module-name. This is followed by one or more sections, in order:

1. The top-level data nodes defined in the module, offset by 4 spaces.

2. Augmentations, offset by 2 spaces and identified by the keyword "augment" followed by the augment target node and a colon (":") character.

3. RPCs, offset by 2 spaces and identified by "rpcs:".

4. Notifications, offset by 2 spaces and identified by "notifications:".

5. Groupings, offset by 2 spaces, and identified by the keyword "grouping" followed by the name of the grouping and a colon (":") character.

6. yang-data, offset by 2 spaces, and identified by the keyword "yang-data" followed by the name of the yang-data structure and a colon (":") character.

The relative organization of each section is provided using a text-based format that is typical of a file system directory tree display command. Each node in the tree is prefaces with "+--". Schema nodes that are children of another node are offset from the parent by 3 spaces. Schema peer nodes separated are listed with the same space offset and, when separated by lines, linked via a vertical bar ("|") character.

The full format, including spacing conventions is:

```
module: <module-name>
```

```
       +--<node>
       |  +--<node>
       |     +--<node>
       +--<node>
          +--<node>
             +--<node>

     augment <target-node>:
       +--<node>
          +--<node>
          +--<node>
             +--<node>
     augment <target-node>:
       +--<node>

     rpcs:
       +--<rpc-node>
       +--<rpc-node>
          +--<node>
          |  +--<node>
          +--<node>

     notifications:
       +--<notification-node>
       +--<notification-node>
          +--<node>
          |  +--<node>
          +--<node>

     grouping <grouping-name>:
       +--<node>
          +--<node>
          |  +--<node>
          +--<node>
     grouping <grouping-name>:
       +--<node>

     yang-data <yang-data-name>:
       +--<node>
          +--<node>
          |  +--<node>
          +--<node>
     yang-data <yang-data-name>:
       +--<node>
```

2.1. Submodules

Submodules are represented in the same fashion as modules, but are identified by "submodule:" followed the (sub)module-name. For example:

```
submodule: <module-name>

  +--<node>
  |  +--<node>
  |     +--<node>
```

2.2. Groupings

Nodes within a used grouping are expanded as if the nodes were defined at the location of the uses statement.

Groupings may optionally be present in the "groupings" section.

2.3. yang-data

If the module defines a "yang-data" structure [RFC8040], these structures may optionally be present in the "yang-data" section.

2.4. Collapsed Node Representation

At times when the composition of the nodes within a module schema are not important in the context of the presented tree, peer nodes and their children can be collapsed using the notation "..." in place of the text lines used to represent the summarized nodes. For example:

```
  +--<node>
  |  ...
  +--<node>
     +--<node>
        +--<node>
```

2.5. Comments

Single line comments, starting with "//" and ending at the end of the line, may be used in the tree notation.

2.6. Node Representation

Each node in a YANG module is printed as:

```
     <status> <flags> <name> <opts> <type> <if-features>

       <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 and actions
         -n  for notifications
         mp  for nodes containing a "mount-point" extension statment

       <name> is the name of the node
         (<name>) means that the node is a choice node
        :(<name>) means that the node is a case 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, choice, anydata or anyxml
         !  for a presence container
         *  for a leaf-list or list
         [<keys>] for a list's keys
         /  for a top-level data node in a mounted module
         @  for a top-level data node in a parent referenced module

       <type> is the name of the type for leafs and leaf-lists

         If the type is a leafref, the type is printed as "-> TARGET",
         where TARGET is either the leafref path, with prefixed removed
         if possible.

       <if-features> is the list of features this node depends on,
         printed within curly brackets and a question mark "{...}?"
```

3. Usage Guidelines For RFCs

This section provides general guidelines related to the use of tree diagrams in RFCs.

3.1. Wrapping Long Lines

Internet Drafts and RFCs limit the number of characters that may in a line of text to 72 characters. When the tree representation of a node results in line being longer than this limit the line should be broken between <opts> and <type>. The type should be indented so that the new line starts below <name> with a white space offset of at least two characters. For example:

```
  notifications:
    +---n yang-library-change
       +--ro module-set-id
               -> /modules-state/module-set-id
```

The previously mentioned "pyang" command can be helpful in producing such output, for example the above example was produced using:

```
  pyang -f tree --tree-line-length 50 ietf-yang-library.yang
```

When a tree diagram is included as a figure in an Internet Draft or RFC, "--tree-line-length 69" works well.

3.2. Long Diagrams

As tree diagrams are intended to provide a simplified view of a module, diagrams longer than a page should generally be avoided. If the complete tree diagram for a module becomes too long, the diagram can be split into several smaller diagrams. For example, it might be possible to have one diagram with the data node and another with all notifications. If the data nodes tree is too long, it is also possible to split the diagram into smaller diagrams for different subtrees. When long diagrams are included in a document, authors should consider whether to include the long diagram in the main body of the document or in an appendix.

An example of such a split can be found in [RFC7407], where section 2.4 shows the diagram for "engine configuration":

```
      +--rw snmp
         +--rw engine
            // more parameters from the "engine" subtree here
```

Further, section 2.5 shows the diagram for "target configuration":

```
      +--rw snmp
         +--rw target* [name]
            // more parameters from the "target" subtree here
```

The previously mentioned "pyang" command can be helpful in producing such output, for example the above example was produced using:

```
pyang -f tree --tree-path /snmp/target ietf-snmp.yang
```

4. YANG Schema Mount Tree Diagrams

YANG Schema Mount is defined in [I-D.ietf-netmod-schema-mount] and warrants some specific discussion. Schema mount is a generic mechanism that allows for mounting of one or more data modules at a specified location of another (parent) schema. The specific location is referred to as a mount point, and any container or list node in a schema may serve as a mount point. Mount points are identified via the inclusion of the "mount-point" extension statement as a substament under a container or list node. Mount point nodes are thus directly identified in a module schema definition and can be identified in a tree diagram as indicated above using the "mp" flag.

In the following example taken from [I-D.ietf-rtgwg-ni-model], "vrf-root" is a container that includes the "mount-point" extension statement as part of its definition:

```
module: ietf-network-instance
  +--rw network-instances
     +--rw network-instance* [name]
        +--rw name           string
        +--rw enabled?       boolean
        +--rw description?   string
        +--rw (ni-type)?
        +--rw (root-type)
           +--:(vrf-root)
           |  +--mp vrf-root
```

4.1. Representation of Instance Data Trees

The actual modules made available under a mount point is controlled by a server and is provided to clients. This information is typically provided via the Schema Mount module defined in [I-D.ietf-netmod-schema-mount]. The Schema Mount module supports exposure of both mounted schema and "parent-references". Parent references are used for XPath evaluation within mounted modules and do not represent client-accessible paths; the referenced information is available to clients via the parent schema. Schema mount also defines an "inline" type mount point where mounted modules are exposed via the YANG library module.

While the modules made available under a mount point are not specified in YANG modules that include mount points, the document defining the module will describe the intended use of the module and may identify both modules that will be mounted and parent modules that can be referenced by mounted modules. An example of such a description can be found in [I-D.ietf-rtgwg-ni-model]. A specific implementation of a module containing mount points will also support a specific list of mounted and referenced modules. In describing both intended use and actual implementations, it is helpful to show how mounted modules would be instantiated and referenced under a mount point using tree diagrams.

In such diagrams, the mount point should be treated much like a container that uses a grouping. The flags should also be set based on the "config" leaf mentioned above, and the mount realted options indicated above should be shown for the top level nodes in a mounted or referenced module. The following example, taken from [I-D.ietf-rtgwg-ni-model], represents the prior example with YANG Routing and OSPF modules mounted, YANG Interface module nodes accessible via a parent-reference, and "config" indicating true:

```
module: ietf-network-instance
  +--rw network-instances
     +--rw network-instance* [name]
        +--rw name           string
        +--rw enabled?       boolean
        +--rw description?   string
        +--rw (ni-type)?
        +--rw (root-type)
           +--:(vrf-root)
              +--mp vrf-root
                 +--ro rt:routing-state/
                 |  +--ro router-id?
                 |  +--ro control-plane-protocols
                 |     +--ro control-plane-protocol* [type name]
                 |        +--ro ospf:ospf
                 |           +--ro instance* [af]
                 |           ...
                 +--rw rt:routing/
                 |  +--rw router-id?
                 |  +--rw control-plane-protocols
                 |     +--rw control-plane-protocol* [type name]
                 |     +--rw ospf:ospf
                 |        +--rw instance* [af]
                 |           ...
                 +--ro if:interfaces@
                 |  ...
                 +--ro if:interfaces-state@
                 |  ...
```

It is worth highlighting that the OSPF module augments the Routing module, and while it is listed in the Schema Mount module (or inline YANG library) there is no special mount-related notation in the tree diagram.

A mount point definition alone is not sufficient to identify if the mounted modules are used for configuration or for non-configuration data. This is determined by the "ietf-yang-schema-mount" module’s "config" leaf associated with the specific mount point and is indicated on the top level mounted nodes. For example in the above tree, when the "config" for the routing module indicates false, the only change would be to the flag on the rt:routing node:

```
                     +--ro rt:routing/
```

5. IANA Considerations

There are no IANA requests or assignments included in this document.

6. Security Considerations

There is no security impact related to the tree diagrams defined in this document.

7. Informative References

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-08 (work in progress), October 2017.

[I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Network Instances", draft-ietf-rtgwg-ni-model-04 (work in progress), September 2017.

[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>.

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <https://www.rfc-editor.org/info/rfc7407>.

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

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

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Lou Berger (editor)
LabN Consulting, L.L.C.

Email: lberger@labn.net

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: August 12, 2018

M. Bjorklund
Tail-f Systems
L. Berger, Ed.
LabN Consulting, L.L.C.

February 8, 2018

YANG Tree Diagrams
draft-ietf-netmod-yang-tree-diagrams-06

Abstract

This document captures the current syntax used in YANG module Tree Diagrams. The purpose of this document is to provide a single location for this definition. This syntax may be updated from time to time based on the evolution of the YANG language.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

YANG Tree Diagrams were first published in [RFC6536]. Such diagrams are used to provided a simplified graphical representation of a data model and can be automatically generated via tools such as "pyang". (See <https://github.com/mbj4668/pyang>). This document describes the syntax used in YANG Tree Diagrams. It is expected that this document will be updated or replaced as changes to the YANG language, see [RFC7950], necessitate.

Today's common practice is to include the definition of the syntax used to represent a YANG module in every document that provides a tree diagram. This practice has several disadvantages and the purpose of this document is to provide a single location for this definition. It is not the intent of this document to restrict future changes, but rather to ensure such changes are easily identified and suitably agreed upon.

An example tree diagram can be found in [RFC7223] Section 3. A portion of which follows:

```
     +--rw interfaces
     |  +--rw interface* [name]
     |     +--rw name                        string
     |     +--rw description?                string
     |     +--rw type                        identityref
     |     +--rw enabled?                    boolean
     |     +--rw link-up-down-trap-enable?   enumeration
```

2. Tree Diagram Syntax

This section describes the meaning of the symbols used in YANG Tree diagrams.

A full tree diagram of a module represents all elements. It includes the name of the module and sections for top level module statements (typically containers), augmentations, rpcs and notifications all identified under a module statement. Module trees may be included in a document as a whole, by one or more sections, or even subsets of nodes.

A module is identified by "module:" followed the module-name. This is followed by one or more sections, in order:

1. The top-level data nodes defined in the module, offset by 2 spaces.

2. Augmentations, offset by 2 spaces and identified by the keyword "augment" followed by the augment target node and a colon (":") character.

3. RPCs, offset by 2 spaces and identified by "rpcs:".

4. Notifications, offset by 2 spaces and identified by "notifications:".

5. Groupings, offset by 2 spaces, and identified by the keyword "grouping" followed by the name of the grouping and a colon (":") character.

6. yang-data, offset by 2 spaces, and identified by the keyword "yang-data" followed by the name of the yang-data structure and a colon (":") character.

The relative organization of each section is provided using a text-based format that is typical of a file system directory tree display command. Each node in the tree is prefaces with "+--". Schema nodes that are children of another node are offset from the parent by 3 spaces. Sibling schema nodes are listed with the same space offset and, when separated by lines, linked via a vertical bar ("|") character.

The full format, including spacing conventions is:

```
     module: <module-name>
       +--<node>
       |  +--<node>
       |     +--<node>
       +--<node>
          +--<node>
             +--<node>

       augment <target-node>:
         +--<node>
            +--<node>
            +--<node>
               +--<node>
       augment <target-node>:
         +--<node>

       rpcs:
         +--<rpc-node>
         +--<rpc-node>
            +--<node>
            |  +--<node>
            +--<node>

       notifications:
         +--<notification-node>
         +--<notification-node>
            +--<node>
            |  +--<node>
            +--<node>

       grouping <grouping-name>:
         +--<node>
            +--<node>
            |  +--<node>
            +--<node>
       grouping <grouping-name>:
         +--<node>

       yang-data <yang-data-name>:
         +--<node>
            +--<node>
            |  +--<node>
            +--<node>
       yang-data <yang-data-name>:
         +--<node>
```

2.1. Submodules

Submodules are represented in the same fashion as modules, but are identified by "submodule:" followed the (sub)module-name. For example:

```
  submodule: <module-name>
    +--<node>
    |  +--<node>
    |     +--<node>
```

2.2. Groupings

Nodes within a used grouping are normally expanded as if the nodes were defined at the location of the "uses" statement. However, it is also possible to not expand the "uses" statement, but instead print the name of the grouping.

For example, the following diagram shows the "tls-transport" grouping from [RFC7407] unexpanded:

```
    +--rw tls
       +---u tls-transport
```

If the grouping is expanded, it could be printed as:

```
    +--rw tls
       +--rw port?                 inet:port-number
       +--rw client-fingerprint?   x509c2n:tls-fingerprint
       +--rw server-fingerprint?   x509c2n:tls-fingerprint
       +--rw server-identity?      snmp:admin-string
```

Groupings may optionally be present in the "groupings" section.

2.3. yang-data

If the module defines a "yang-data" structure [RFC8040], these structures may optionally be present in the "yang-data" section.

2.4. Collapsed Node Representation

At times when the composition of the nodes within a module schema are not important in the context of the presented tree, sibling nodes and their children can be collapsed using the notation "..." in place of the text lines used to represent the summarized nodes. For example:

```
       +--<node>
       |  ...
       +--<node>
          +--<node>
             +--<node>
```

2.5. Comments

Single line comments, starting with "//" (possibly indented) and ending at the end of the line, may be used in the tree notation.

2.6. Node Representation

Each node in a YANG module is printed as:

```
     <status>--<flags> <name><opts> <type> <if-features>

       <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, output parameters to rpcs
             and actions, and notification parameters
         -w  for input parameters to rpcs and actions
         -u  for uses of a grouping
         -x  for rpcs and actions
         -n  for notifications
         mp  for nodes containing a "mount-point" extension statement

       <name> is the name of the node
         (<name>) means that the node is a choice node
        :(<name>) means that the node is a case node

         If the node is augmented into the tree from another module,
```

its name is printed as <prefix>:<name>, where <prefix> is the prefix defined in the module where the node is defined.

<opts> is one of:

```
  ?  for an optional leaf, choice, anydata or anyxml
  !  for a presence container
  *  for a leaf-list or list
  [<keys>] for a list's keys
  /  for a top-level data node in a mounted module
  @  for a top-level data node in a parent referenced module
```

<type> is the name of the type for leafs and leaf-lists

If the type is a leafref, the type is either printed as "-> TARGET", where TARGET is the leafref path, with prefixes removed if possible, or printed as "leafref".

<if-features> is the list of features this node depends on, printed within curly brackets and a question mark "{...}?"

Arbitrary whitespace is allowed between any of the whitespace separated fields (e.g., <opts> and <type>). Additional whitespace may for example be used to column align fields (e.g., within a list or container) to improve readability.

3. Usage Guidelines For RFCs

This section provides general guidelines related to the use of tree diagrams in RFCs.

3.1. Wrapping Long Lines

Internet Drafts and RFCs limit the number of characters that may in a line of text to 72 characters. When the tree representation of a node results in line being longer than this limit the line should be broken between <opts> and <type>, or between <type> and <if-feature>. The new line should be indented so that it starts below <name> with a white space offset of at least two characters. For example:

```
  notifications:
    +---n yang-library-change
       +--ro module-set-id
               -> /modules-state/module-set-id
```

Long paths (e.g., leafref paths or augment targets) can be split and printed on more than one line. For example:

```
  augment /nat:nat/nat:instances/nat:instance/nat:mapping-table
            /nat:mapping-entry:
```

The previously mentioned "pyang" command can be helpful in producing such output, for example the notification diagram above was produced using:

```
  pyang -f tree --tree-line-length 50 ietf-yang-library.yang
```

When a tree diagram is included as a figure in an Internet Draft or RFC, "--tree-line-length 69" works well.

3.2. Groupings

If the YANG module is comprised of groupings only, then the tree diagram should contain the groupings. The 'pyang' compiler can be used to produce a tree diagram with groupings using the "-f tree --tree-print-groupings" command line parameters.

3.3. Long Diagrams

Tree diagrams can be split into sections to correspond to document structure. As tree diagrams are intended to provide a simplified view of a module, diagrams longer than a page should generally be avoided. If the complete tree diagram for a module becomes too long, the diagram can be split into several smaller diagrams. For example, it might be possible to have one diagram with the data node and another with all notifications. If the data nodes tree is too long, it is also possible to split the diagram into smaller diagrams for different subtrees. When long diagrams are included in a document, authors should consider whether to include the long diagram in the main body of the document or in an appendix.

An example of such a split can be found in [RFC7407], where section 2.4 shows the diagram for "engine configuration":

```
    +--rw snmp
       +--rw engine
          // more parameters from the "engine" subtree here
```

Further, section 2.5 shows the diagram for "target configuration":

```
    +--rw snmp
       +--rw target* [name]
          // more parameters from the "target" subtree here
```

The previously mentioned "pyang" command can be helpful in producing such output, for example the above example was produced using:

```
pyang -f tree --tree-path /snmp/target ietf-snmp.yang
```

4. YANG Schema Mount Tree Diagrams

YANG Schema Mount is defined in [I-D.ietf-netmod-schema-mount] and warrants some specific discussion. Schema mount is a generic mechanism that allows for mounting of one or more YANG modules at a specified location of another (parent) schema. The specific location is referred to as a mount point, and any container or list node in a schema may serve as a mount point. Mount points are identified via the inclusion of the "mount-point" extension statement as a substatement under a container or list node. Mount point nodes are thus directly identified in a module schema definition and can be identified in a tree diagram as indicated above using the "mp" flag.

In the following example taken from [I-D.ietf-rtgwg-ni-model], "vrf-root" is a container that includes the "mount-point" extension statement as part of its definition:

```
module: ietf-network-instance
  +--rw network-instances
     +--rw network-instance* [name]
        +--rw name           string
        +--rw enabled?       boolean
        +--rw description?   string
        +--rw (ni-type)?
        +--rw (root-type)
           +--:(vrf-root)
           |  +--mp vrf-root
```

4.1. Representation of Mounted Schema Trees

The actual modules made available under a mount point is controlled by a server and is provided to clients. This information is typically provided via the Schema Mount module defined in [I-D.ietf-netmod-schema-mount]. The Schema Mount module supports exposure of both mounted schema and "parent-references". Parent references are used for XPath evaluation within mounted modules and do not represent client-accessible paths; the referenced information is available to clients via the parent schema. Schema mount also defines an "inline" type mount point where mounted modules are exposed via the YANG library module.

While the modules made available under a mount point are not specified in YANG modules that include mount points, the document defining the module will describe the intended use of the module and may identify both modules that will be mounted and parent modules that can be referenced by mounted modules. An example of such a description can be found in [I-D.ietf-rtgwg-ni-model]. A specific implementation of a module containing mount points will also support a specific list of mounted and referenced modules. In describing both intended use and actual implementations, it is helpful to show how mounted modules would be instantiated and referenced under a mount point using tree diagrams.

In such diagrams, the mount point should be treated much like a container that uses a grouping. The flags should also be set based on the "config" leaf mentioned above, and the mount related options indicated above should be shown for the top level nodes in a mounted or referenced module. The following example, taken from [I-D.ietf-rtgwg-ni-model], represents the prior example with YANG Routing and OSPF modules mounted, YANG Interface module nodes accessible via a parent-reference, and "config" indicating true:

```
module: ietf-network-instance
  +--rw network-instances
     +--rw network-instance* [name]
        +--rw name           string
        +--rw enabled?       boolean
        +--rw description?   string
        +--rw (ni-type)?
        +--rw (root-type)
           +--:(vrf-root)
              +--mp vrf-root
                 +--ro rt:routing-state/
                 |  +--ro router-id?
                 |  +--ro control-plane-protocols
                 |     +--ro control-plane-protocol* [type name]
                 |        +--ro ospf:ospf
                 |           +--ro instance* [af]
                 |           ...
                 +--rw rt:routing/
                 |  +--rw router-id?
                 |  +--rw control-plane-protocols
                 |     +--rw control-plane-protocol* [type name]
                 |     +--rw ospf:ospf
                 |        +--rw instance* [af]
                 |           ...
                 +--ro if:interfaces@
                 |  ...
                 +--ro if:interfaces-state@
                 |  ...
```

It is worth highlighting that the OSPF module augments the Routing module, and while it is listed in the Schema Mount module (or inline

YANG library) there is no special mount-related notation in the tree diagram.

A mount point definition alone is not sufficient to identify if the mounted modules are used for configuration or for non-configuration data. This is determined by the "ietf-yang-schema-mount" module's "config" leaf associated with the specific mount point and is indicated on the top level mounted nodes. For example in the above tree, when the "config" for the routing module indicates false, the nodes in the "rt:routing" subtree would have different flags:

```
+--ro rt:routing/
|  +--ro router-id?
|  +--ro control-plane-protocols
   ...
```

5. IANA Considerations

There are no IANA requests or assignments included in this document.

6. Security Considerations

There is no security impact related to the tree diagrams defined in this document.

7. Informative References

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-08 (work in progress), October 2017.

[I-D.ietf-rtgwg-ni-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Network Instances", draft-ietf-rtgwg-ni-model-05 (work in progress), December 2017.

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

[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>.

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <https://www.rfc-editor.org/info/rfc7407>.

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

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

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Lou Berger (editor)
LabN Consulting, L.L.C.

Email: lberger@labn.net

Network Working Group   C. Hopps
Internet-Draft   Deutsche Telekom
Updates: rfc6087bis (if approved)   L. Berger
Intended status: Standards Track   LabN Consulting, L.L.C.
Expires: April 27, 2018   D. Bogdanovic
                                                        October 24, 2017

YANG Module Tags
draft-rtgyangdt-netmod-module-tags-02

Abstract

This document provides for the association of tags with YANG modules. The expectation is for such tags to be used to help classify and organize modules. A method for defining, reading and writing a modules tags is provided, as well as an augmentation to YANG library. Tags may be standardized and assigned during module definition; assigned by implementations; or dynamically defined and set by users. This document provides guidance to future model writers and, as such, this document updates [I-D.ietf-netmod-rfc6087bis].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The use of tags for classification and organization is fairly ubiquitous not only within IETF protocols, but in the internet itself (e.g., #hashtags). Tags can be usefully standardized, but they can also serve as a non-standardized mechanism available for users to define themselves. Our solution provides for both cases allowing for the most flexibility. In particular, tags may be standardized as well as assigned during module definition; assigned by implementations; or dynamically defined and set by users.

This document defines two modules. The first module defines a list of module entries to allow for adding or removing of tags. It also defines an RPC to reset a modules tags to the original values. The second module defines an augmentation to YANG Library [RFC7895] to allow for reading a modules tags.

This document also defines an IANA registry for tag prefixes as well as a set of globally assigned tags.

Section 9 provides guidelines for authors of YANG data models. This section updates [I-D.ietf-netmod-rfc6087bis].

2. Conventions Used in This Document

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

Note that lower case versions of these key words are used in section Section 9 where guidance is provided to future document authors.

3. Tag Locations

Each module has only one logical tag list; however, that tag list may be accessed from multiple locations.

We define two tag list locations. The first location is used for configuration and is a top level list of module entries where each entry contain the list of tags. The second read-only location is through the yang library under the module entry.

4. Tag Prefixes

All tags have a prefix indicating who owns their definition. An IANA registry is used to support standardizing tag prefixes. Currently 3 prefixes are defined with all others reserved.

4.1. IETF Standard Tags

An IETF standard tag is a tag that has the prefix "ietf:". All IETF standard tags are registered with IANA in a registry defined later in this document.

4.2. Vendor Tags

A vendor tag is a tag that has the prefix "vendor:". These tags are defined by the vendor that implements the module, and are not standardized; however, it is recommended that the vendor consider including extra identification in the tag name to avoid collisions (e.g., vendor:super-duper-company:...).

4.3. Local Tags

A local tag is any tag that has the prefix "local:". These tags are defined by the local user/administrator and will never be standardized.

4.4. Reserved Tags

Any tag not starting with the prefix "ietf:", "vendor:" or "local:" is reserved for future standardization.

5. Tag Management

Tags can become associated with a module in a number of ways. Tags may be defined and associated at model design time, at implementation time, or via user administrative control. As the main consumer of tags are users, users may also remove any tag, no matter how the tag became associated with a module.

5.1. Module Definition Association

A module definition SHOULD indicate a set of tags to be automatically added by the module implementer. These tags MUST be standard tags (Section 4.1). This does imply that new modules may also drive the addition of new standard tags to the IANA registry.

5.2. Implementation Association

An implementation MAY include additional tags associated with a module. These tags may be standard or vendor specific tags.

5.3. Administrative Tagging

Tags of any kind can be assigned and removed with normal configuration mechanisms. Additionally we define an RPC to reset a module’s tag list to the implementation default.

Implementations MUST ensure that a modules tag list is consistent across any location from which the list is accessible. So if a user adds a tag through configuration that tag should also be seen when using the yang library augmentation.

Implementations that do not support the reset rpc statement (whether at all, or just for a particular rpc or module) MUST respond with an

YANG transport protocol-appropriate rpc layer error when such a statement is received.

5.3.1. Resetting Tags

The "reset-tags" rpc statement is defined to reset a module's tag list to the implementation default, i.e. the tags that are present based on module definition and any that are added during implementation time. This rpc statement takes module identification information as input, and provides the list of tags that are present after the reset.

6. Tags Module Structure

6.1. Tags Module Tree

The tree associated with the tags module is:

```
module: ietf-module-tags
  rpcs:
    +---x reset-tags
       +---w input
       |  +---w name         yang:yang-identifier
       |  +---w revision?    union
       +--ro output
          +--ro tags*    string
```

6.2. Tags Module

```
<CODE BEGINS> file "ietf-module-tags@2017-10-25.yang"
module ietf-module-tags {
  yang-version "1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-module-tags";
  prefix "mtags";

  import ietf-yang-types {
    prefix yang;
  }

  import ietf-yang-library {
    prefix yanglib;
  }

  // meta
  organization "IETF NetMod Working Group (NetMod)";

  contact
    "NetMod Working Group - <netmod@ietf.org>";
```

```
    description
      "This module describes a tagging mechanism for yang module.
       Tags may be IANA assigned or privately defined types.";

    revision "2017-10-25" {
      description
        "Initial revision.";
      reference "TBD";
    }

    grouping module-tags {
      description
        "A grouping that may be used to classify a module.";

      leaf-list tags {
        type string;

        description
          "The module associated tags. See the IANA 'YANG Module Tag
          Prefix' registry for reserved prefixes and the IANA 'YANG
          Module IETF Tag' registry for IETF standard tags";
      }
    }

    grouping yanglib-common-leafs {
      description
        "Common parameters for YANG modules and submodules.
         This definition extract from RFC7895 as it is defined as
         a grouping within a grouping.

         TBD is there a legal way to use a grouping defined within
         another grouping without using the parent? If so, should change
         to that.";

      leaf name {
        type yang:yang-identifier;
        mandatory true;
        description
          "The YANG module or submodule name.";
      }
      leaf revision {
        type union {
          type yanglib:revision-identifier;
          type string { length 0; }
        }
        mandatory true;
        description
          "The YANG module or submodule revision date.
```

```
           A zero-length string is used if no revision statement
           is present in the YANG module or submodule.";
      }
    }

    list module-tags {
      key "name revision";
      description
        "A list of modules and their tags";
      uses yanglib-common-leafs; // uses yanglib:common-leafs;
      uses module-tags;
    }

    rpc reset-tags {
      description
        "Reset a list of tags for a given module to the list of module
         and implementation time defiend tags. It provides the list of
         tags associated with the module post reset.";

      input {
        uses yanglib-common-leafs; // uses yanglib:common-leafs;
      }

      output {
         uses module-tags;
      }
    }
  }
  <CODE ENDS>
```

7. Library Augmentation

A modules tags can also be read using the yang library [RFC7895] if a server supports both YANG library and the augmentation defined below. If a server supports ietf-module-tags and the YANG library it SHOULD also support the ietf-library-tags module.

The tree associated with the defined augmentation is:

```
module: ietf-library-tags
  augment /yanglib:modules-state/yanglib:module:
    +--ro tags*   string
```

7.1. Library Augmentation Module

```
<CODE BEGINS> file "ietf-library-tags@2017-08-12.yang"
module ietf-library-tags {
  // namespace
  namespace "urn:ietf:params:xml:ns:yang:ietf-library-tags";

  prefix ylibtags;

  import ietf-yang-library {
     prefix yanglib;
  }
  import ietf-module-tags {
     prefix mtags;
  }

  // meta
  organization "IETF NetMod Working Group (NetMod)";

  contact
      "NetMod Working Group - <netmod@ietf.org>";

  description
    "This module augments ietf-yang-library with searchable
    classfication tags.  Tags may be IANA or privately defined
    types.";

  revision "2017-08-12" {
    description
      "Initial revision.";
    reference "RFC TBD";
  }

  augment "/yanglib:modules-state/yanglib:module" {
    description
      "The yang library structure is augmented with a module tags
       list. This allows operators to tag modules regardless of
       whether the modules included tag support or not";

    uses mtags:module-tags;

  }
}
<CODE ENDS>
```

8. Other Classifications

It’s worth noting that a different yang module classification document exists [RFC8199]. That document is classifying modules in only a logical manner and does not define tagging or any other mechanisms. It divides yang modules into 2 categories (service or element) and then into one of 3 origins: standard, vendor or user. It does provide a good way to discuss and identify modules in general. This document defines standard tags to support [RFC8199] style classification.

9. Guidelines to Model Writers

This section updates [I-D.ietf-netmod-rfc6087bis].

9.1. Define Standard Tags

A module SHOULD indicate, in the description statement of the module, a set of tags that are to be associated with it. This description should also include the appropriate conformance statement or statements, using [RFC2119] language for each tag.

```
      module sample-module {
        ...
        description
          "[Text describing the module...]

          RFC<this document> TAGS:
          The following tags MUST be included by an implementation:
               - ietf:some-required-tag:foo
               - ...
          The following tags SHOULD be included by an implementation:
               - ietf:some-recommended-tag:bar
               - ...
          The following tags MAY be included by an implementation:
               - ietf:some-optional-tag:baz
               - ...
            ";
        ...
      }
```

One SHOULD only include conformance text if there will be tags listed (i.e., there’s no need to indicate an empty set).

The module writer may use existing standard tags, or use new tags defined in the model definition, as appropriate. New tags should be assigned in the IANA registry defined below, see Section 10.2 below.

10. IANA Considerations

10.1. YANG Module Tag Prefix Registry

This registry allocates tag prefixes. All YANG module tags SHOULD begin with one of the prefixes in this registry.

The allocation policy for this registry is Specification Required [RFC5226].

The initial values for this registry are as follows.

prefix	description
ietf:	IETF Standard Tag allocated in the IANA YANG Module IETF Tag Registry.
vendor:	Non-standardized tags allocated by the module implementer.
local:	Non-standardized tags allocated by and for the user.

Other SDOs (standard organizations) wishing to standardize their own set of tags could allocate a top level prefix from this registry.

10.2. YANG Module IETF Tag Registry

This registry allocates prefixes that have the standard prefix "ietf:". New values should be well considered and not achievable through a combination of already existing standard tags.

The allocation policy for this registry is IETF Review [RFC5226].

The initial values for this registry are as follows.

[Editor’s note: many of these tags may move to [I-D.ietf-rtgwg-device-model] if/when that document is refactored to use tags.]

Tag	Description	Reference
ietf:rfc8199:element	A module for a network element.	[RFC8199]
ietf:rfc8199:service	A module for a network service.	[RFC8199]
ietf:rfc8199:standard	A module defined by a standards organization.	[RFC8199]

ietf:rfc8199:vendor	A module defined by a vendor.	[RFC8199]
ietf:rfc8199:user	A module defined by the user.	[RFC8199]
ietf:device:hardware	A module relating to device hardware (e.g., inventory).	[This document]
ietf:device:software	A module relating to device software (e.g., installed OS).	[This document]
ietf:device:qos	A module for managing quality of service.	[This document]
ietf:protocol	A module representing a protocol.	[This document]
ietf:system-management	A module relating to system management (e.g., a system management protocol).	[This document]
ietf:network-service	A module relating to network service (e.g., a network service protocol).	[This document]
ietf:oam	A module representing Operations, Administration, and Maintenance.	[This document]
ietf:routing	A module related to routing.	[This document]
ietf:routing:rib	A module related to routing information bases.	[This document]
ietf:routing:igp	An interior gateway protocol module.	[This document]
ietf:routing:egp	An exterior gateway protocol module.	[This document]
ietf:signaling	A module representing control plane signaling.	[This document]
ietf:lmp	A module representing a link management protocol.	[This document]

+----------------------+--------------------------+----------+

Table 1: IETF Module Tag Registry

11. References

11.1. Normative References

[I-D.ietf-netmod-rfc6087bis]
Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", draft-ietf-netmod-rfc6087bis-14 (work in progress), September 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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

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

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

11.2. Informative References

[I-D.ietf-rtgwg-device-model]
Lindem, A., Berger, L., Bogdanovic, D., and C. Hopps, "Network Device YANG Logical Organization", draft-ietf-rtgwg-device-model-02 (work in progress), March 2017.

Authors’ Addresses

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Dean Bogdanovic

Email: ivandean@gmail.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

N. Sambo
P. Castoldi
Scuola Superiore Sant'Anna
G. Fioccola
Telecom Italia
F. Cugini
CNIT
H. Song
T. Zhou
Huawei

October 30, 2017

YANG model for finite state machine
draft-sambo-netmod-yang-fsm-00

Abstract

Network operators and service providers are facing the challenge of deploying systems from different vendors while looking for a trade-off among transmission performance, network device reuse, and capital expenditure without the need of being tied to single vendor equipment. The deployment and operation of more dynamic and programmable network infrastructures can be driven by adopting model-driven and software-defined control and management paradigms. In this context, YANG enables to compile a set of consistent vendor-neutral data models for networks and components based on actual operational needs emerging from heterogeneous use cases. This document proposes YANG models to describe events, operations, and finite state machine of YANG-defined network elements. The proposed models can be applied in several use cases: i) in the context of optical networks to pre-instruct data plane devices (e.g., an optical transponder) on the actions to be performed (e.g., code adaptation) in case some events, such as physical layer degradations, occur; ii) in general data networks, network telemetry applications can define and embed custom data probes into data plane devices. A probe in many cases can be modeled as an FSM; iii) the monitoring of packet loss and delay through a network clustering approach.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Networks are evolving toward more programmability, flexibility, and multi-vendor interoperability. Multi-vendor interoperability can be applied in the context of nodes, i.e. a node composed of components provided by different vendors (named fully disaggregated white box) is assembled under the same control system. This way, operators can optimize costs and network performance without the need of being tied to single vendor equipment. NETCONF protocol RFC6241 [RFC6241] based on YANG data modeling language RFC6020 [RFC6020] is emerging as a candidate Software Defined Networking (SDN) enabled protocol. First, NETCONF supports both control and management functionalities, thus permits high programmability. Then, YANG enables data modeling in a vendor-neutral way. Some recent works have provided YANG models to describe attributes of links (e.g., identification), nodes (e.g., connectivity matrix), media channels, and transponders (e.g., supported forward error correction - FEC) of networks ([I-D.ietf-i2rs-yang-network-topo] [I-D.vergara-ccamp-flexigrid-yang] [I-D.zhang-ccamp-l1-topo-yang]), also including optical technologies. This document presents YANG models to describe events, operations, and finite state machine of YANG-defined network elements. Such models can be applied to several use cases. In the context of elastic optical networks (EONs), the model enables a centralized remote network controller (managed by a network operator) to instruct a transponder controller about the actions to perform when certain events (e.g., failures) occur. The actions to be taken and the events can be re-programmed on the device. In general data networks, programmable network telemetry is considered a killer SDN application which can help applications gain unprecedented visibility to network data plane. Instead of providing raw data, network devices can be configured to filter and process data directly on the data plane and only hand preprocessed data to the collector, in order to save data bandwidth and reduce reaction delay ([I-D.song-opsawg-dnp4iq]) . Such configurations can be programed as custom probes and dynamically deployed into data plane devices. A probe in many cases can be modeled as an FSM. Another use case is the monitoring of packet loss and delay through a network clustering approach: in this case, each FSM state is determined by a specific subdivision of the network in Clusters ([I-D.fioccola-ippm-multipoint-alt-mark]).

2. Conventions used in this document

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

3. Terminology

ABNO: Application-Based Network Operations

BER: Bit Error Rate

EON: Elastic Optical Network

FEC: Forward Error Correction

FSM: Finite State Machine

NETCONF: Network Configuration Protocol

OAM: Operation Administration and Maintenance

SDN: Software Defined Network

YANG: Yet Another Network Generator

DNP: Dynamic Network Probe

AMM: Alternate Marking Method

4. Example of application

4.1. Pre-programming resiliency schemes in EONs

EONs (optical networks based on flexible grid supporting circuits of different bandwidth) are expected to employ flexible transponders, i.e. transponders supporting multiple bit rates, multiple modulation formats, and multiple codes. Such transponders permits the (re-) configuration of the bit rate value based on traffic requirements, as well as the configuration of the modulation format and code based on the physical characteristics of a path (e.g., quadrature phase shift keying is more robust than 16 quadrature amplitude modulation). This way, transmission parameters can be (re-) configured based on physical layer changes. The YANG model presented in this draft enables to pre-program reconfiguration settings of data plane devices in case of failures or physical layer degradations. In particular, soft failures are assumed. Soft failures imply transmission performance degradation, in turns a bit error rate (BER) increase, e.g. due to the ageing of some network devices. Without loosing generality, the ABNO architecture is assumed for the control and management of EONs (RFC7491 [RFC7491]). Considering the state of the art, when pre-FEC BER passes above a predefined threshold, it is expected that an alarm is sent to the OAM Handler, which communicates with the ABNO controller that may trigger an SDN controller (that could be the Provisioning Manager of ABNO RFC7491 [RFC7491]) for computing new transmission parameters. The involved ABNO modules are shown in the simplified ABNO architecture of Fig. 1. Then, transponders are reconfigured. When alarms related to several connections impacted by the soft failure are generated, this procedure may be particularly time consuming. The related workflow for transponder reconfiguration is shown in Fig. 2. The proposed model enables an SDN controller to instruct the transponder about reconfiguration of new transmission parameters values if a soft failure occurs. This can be done before the failure occurs (e.g., during the connection instantiation phase or during the connection service), so that data plane devices can promptly reconfigure themselves without querying the SDN controller to trigger an on-demand recovery. This is expected to speed up the recovery process from soft failures. The related flow chart is shown in Fig. 3.

```
          ___________            ___________
         |  ABNO      |         |   OAM      |
         |controller |  ------  |  Handler   |
         |___________|          |___________|

           |                         |
           |                         |
           |                         |
       ____________                  |
      |    SDN      |                |
      | controller |                 |
      |____________|                 |
                                     |
            |                        |
            |                        |
            |                        |
          _____________________________
         |            Client            |
         |            network           |
         |_____________________________|
```

Figure 1: Assumed ABNO functional modules

```
                  _____________________
                 |         1           |
                 |Sending alarm to the |
                 |     OAM Handler     |
                 |                     |
                 |_____________________|
                           |
                           |
                           |
                 _____________________
                |         2           |
                |      Trigger        |
                |   SDN Controller    |
                |                     |
                |_____________________|
                           |
                           |
                           |
                 _____________________
                |         3           |
                |  Computation of     |
                | new transmission    |
                |   parameters        |
                |_____________________|
                          |
                          |
                          |
                 _____________________
                |        4            |
                |   Data plane        |
                |   reconfiguration   |
                |                     |
                |_____________________|
```

Figure 2: Flow chart of the expected state-of-the-art approach

```
             _______________________
            |           1           |
            | Instructing the local |
            |     controller of     |
            |  data plane devices   |
            |_______________________|
                        |
                        |
                        |
             _______________________
            |           2           |
            | Local reconfiguration |
            |     upon failure      |
            |       detection       |
            |_______________________|
                        |
                        |
                        |
             _______________________
            |           3           |
            |                       |
            |     notification      |
            |                       |
            |_______________________|
```

Figure 3: Flow chart of the approach exploiting YANG models in this draft

4.2. Deploying Dynamic Probes for Programmable Network Telemetry

In the past, network data analytics was considered a separate function from networks. They consume raw data extracted from networks through piecemeal protocols and interfaces. With the advent of user programmable data plane, we expect a paradigm shift that makes the data plane be an active component of the data telemetry and analytics solution. The programmable in-network data preprocessing is efficient and flexible to offload some light-weight data processing through dynamic data plane programming or configuration. A universal network data analytics platform built on top of this enables a tight and agile network control and OAM feedback loop. A proposed dynamic network telemetry system architecture is illustrated in Fig.4.

An application translates its data requirements into a set of Dynamic Network Probes (DNP) targeting a subset of data plane devices. After the probes are deployed, each probe conducts its corresponding in-network data preprocessing and feeds the preprocessed data to the collector. The collector finishes the data post-processing and presents the results to the data-requesting application.

```
+--------------------------------------+
|  network telemetry applications      |
+--------------------------------------+
       ^                    |
       |                    V
       |          +--------------------+
       |          | DNP compile/config |
       |          +--------------------+
       |                    |
       |                    V
+---------------+ +--------------------+
|data collection| | Probe deployment   |
+---------------+ +--------------------+
    ^   ^   ^            |   |   |
    |   |   |            V   V   V
+---------------------------------------+
|  network data plane devices           |
| (in-network data preprocessing)       |
+---------------------------------------+
```

Figure 4: Deploy dynamic network probes using YANG FSM models

Many DNPs can be modeled as FSM which are configured to capture specific events. Here FSMs essentially preprocess the raw stream data and only report the necessary data to subscribing applications.

For example, a congestion control application needs to monitor the router buffer occupancy. Instead of polling the buffer depth periodically, it is only interested in the real-time events when the buffer depth crosses a low and a high threshold. We can install a probe to achieve this data plane function and the probe can be modeled as a three-state FSM. Each state represents a buffer region: below the low threshold, above the high threshold, and in between the two thresholds. A possible state transition is checked against the buffer depth for each incoming and outgoing packet. Whenever a state transition happens, an event is generated and reported to the application. This approach significantly reduces the amount of data sent to the application and also allows a timely event notification.

For another example, an application would like to monitor the delay experienced by a flow. The packet delay on its forwarding path can be acquired by using iOAM [I-D.brockners-inband-oam-requirements]. However, the application only needs to know that N consecutive flow packets experience a delay longer than T. Instead of forwarding the raw delay data to the application, a probe can be deployed to detect the event. Similarly, the probe can be modeled as an FSM.

4.3. IP Performance Measurements on multipoint-to-multipoint large Networks

Networks offer rich sets of network performance measurement data, but traditional approaches run into limitations. One reason for this is the fact that in many cases, the bottleneck is the generation and export of the data and the amount of data that can be reasonably collected from the network runs into bandwidth and processing constraints in the network itself. In addition, management tasks related to determining and configuring which data to generate lead to significant deployment challenges.

In order to address these issues, an SDN controller application orchestrates network performance measurements tasks across the network to allow an optimized monitoring. In fact the IP Performance Measurement SDN Controller Application in Figure 5 can calibrate how deep can be obtained monitoring data from the network by configuring measurement points roughly or meticulously. This can be established by using the feedback mechanism reported in Figure 5.

For instance, the SDN Controller can configure initially an end to end monitoring between ingress points and egress points of the network. If the network does not experiment issues, this approximate monitoring is good enough and is very cheap in terms of network resources. But, in case of problems, the SDN Controller becomes aware of the issues from this approximate monitoring and, in order to localize the portion of the network that has issues, configures the measurement points more exhaustively. So a new detailed monitoring is performed. After the detection and resolution of the problem the initial approximate monitoring can be used again. This idea is general and can be applied to different performance measurements techniques both active and passive (and hybrid).

```
+---------------------------------------+
|       IP Performance Measurement      |
|       SDN Controller Application      |
+---------------------------------------+
    ^   ^   ^            |   |   |
    |   |   |            v   v   v
+---------------------------------------+
|           Multipoint Network          |
+---------------------------------------+
```

Figure 5: Feedback mechanism on multipoint-to-multipoint large Networks

One of the most efficient methodology to perform packet, loss delay and jitter measurements both in an IP and Overlay Networks is the Alternate Marking method, as presented in [I-D.ietf-ippm-alt-mark] and [I-D.fioccola-ippm-multipoint-alt-mark].

This technique can be applied to point-to-point flows but also to multipoint.to-multipoint flows (see [I-D.ietf-ippm-alt-mark] and [I-D.fioccola-ippm-multipoint-alt-mark]). The Alternate Marking method creates batches of packets by alternating the value of 1 or 2 bits of the packet header. These batches of packets are unambiguously recognized over the network and the comparison of packet counters permits the packet loss calculation. The same idea can be applied for delay measurement by selecting special packets with a marking bit dedicated for delay measurements. This method needs two counters each marking period for each flow under monitor. For this reason by considering n measurement points and n monitored flows, the order of magnitude of the packet counters for each time interval is n*n*2 (1 per color).

Multipoint Alternate Marking, described in [I-D.fioccola-ippm-multipoint-alt-mark], aims to reduce this value and makes the performance monitoring more flexible in case a detailed analysis is not needed.

It is possible to monitor a Multipoint Network without examining in depth by using the Network Clustering (subnetworks that are portions of the entire network that preserve the same property of the entire network). So in case there is packet loss or the delay is too high the filtering criteria could be specified more in order to perform a per flow detailed analysis, as described in [I-D.ietf-ippm-alt-mark].

An application of the multipoint performance monitoring can be done by using FSM (each state is a composition of clusters) and feedback mechanism where the SDN Controller is the brain of the network and can manage flow control to the switches and routers and, in the same way, can calibrate the performance measurements depending on the necessity.

5. YANG for finite state machine (FSM)

This model defines a list of states and transitions to describe a generic finite state machine (FSM). The related code and tree are shown in the Appendix.

```
 <current-state>: it defines the current state of the FSM.
 <states>: this element defines the FSM as follows.
          <state>: this list defines all the FSM states.
                 <id>: this leaf attribute of <state> defines the
```

identifier of the state
<name>: this leaf attribute of <state> defines the name of the state
<description>: this leaf is a "string" describing the state
<transitions>: this attribute defines a list of transitions to other states in the FSM.

<name>: this attribute defines the name of a transition
<type>: this attribute defines the type of the transition from a pool of possible transition types predefined inside the YANG model. Together with the <name> attribute, it uniquely identifies the transition.
<description>: this optional attribute is a "string" describing the transition
<filters>: this leaf is a list of input parameters related to the transition. This attribute enables to further express a transition: as an example, if a transition can be triggered by a parameter (e.g., a monitored performance parameter) exceeding a threshold (as in Sec. 5), an element of the list defines this threshold. Thus, if the parameter is outside the threshold, the transition is taken, otherwise not.

<filter>: this leaf of <filters> defines a filter parameter.
<filter-id>: this leaf of <filters> define the identifier number associated with the <filter> attribute.

<actions>: this attribute defines a list of actions to take during the transition.

<action>: this attribute is the list of actions

<id>: this leaf of <action> defines the identifier number of an action.
<type>: this leaf of <action> defines the type of an action.
<simple>: this leaf defines (differently from <conditional> detailed below) an action that has to be directly executed.

<execute>: this attribute recalls an RPC encapsulating the effective task (action) to be executed by the

hardware. If more actions (e.g., "A" and "B"), defined in the <action> list, have to be executed, these actions can be executed sequentially according to the <next-action> attribute detailed below. Thus, by referring to the tree of the Appendix, when an action ("A") is executed, the <next-action> attribute will bring to another action ("B"). If more actions have to be executed in parallel (e.g., "A" & "B"), not sequentially, an element of the <action> list should be defined to express an action (e.g., "A&B") consisting of more actions to be executed in parallel.

<next-action>: this attribute defines the identification number of a next action that has to be taken. The <next-action> can assume a NULL value.

<conditional>: this leaf enables a check ("true" or "false") to be verified before executing the action. Based on the check, the proper attributes <execute> and <next-operation> are considered.

<statement>: this leaf of <conditional> defines the condition to be verified before executing the action.

<true>: this leaf of <conditional> defines a result of the check associated to <statement>. Proper <execute> and <next-operation> attributes are associated with this result of the

```
                                                  check.
                                                  <false>: this leaf of
                                                  <conditional> defines a
                                                  result of the check
                                                  associated to
                                                  <statement>. Proper
                                                  <execute> and
                                                  <next-operation>
                                                  attributes are associated
                                                  with this result of the
                                                  check.
                                    <next-state>: this attribute defines
                                    the next state of FSM when an action is
                                    executed.
```

6. Implementation of the pre-programming resiliency schemes in EONs

These presented model can be used to enable a centralized network controller, managed by a network operator, to instruct data plane hardware on its reconfiguration if some events, such as a failure or physical layer degradation, occur. As an example, an optical signal impacted by a soft failure (i.e., a physical layer degradation inducing a pre forward error correction bit error rate increase - pre-FEC) can be maintained by adapting the FEC of the signal itself. This action to be taken and, more in general operations to be executed depending on critical events, can be (re-) programmed on the transponder by (re-) sending a NETCONF <edit-config> message to the device controller including a FSM defined by the YANG model. Such a system has the main goal to speed up the reaction of the network to certain events/faults and to alleviate the workload of the centralized controller. The speed up derives from the fact that the centralized controller is able to pre-compute and pre-configure on the network devices the actions to take when an event occurs taking into account a global view and knowledge of the network. In this way, the device is already aware of the actions to be locally applied to reconfigure a connection, avoiding to inform the controller and to wait for the response indicating what to do. Consequently, part of the workload is also removed from the centralized controller. When the reaction is successfully completed in the data plane, the centralized controller can be notified about the faults and the taken action. A flexible transponder supporting two FEC types, 7% and 20%, is considered. A two-states FSM is also assumed. The states have <name> attribute set to "Steady" and "Fec-Baud-Adapt", respectively. In the "Steady" state, the signal is in a healthy condition, adopting a 7% FEC, with a pre-FEC BER below an assigned threshold of 9 x 10-4. A transition from this state can be triggered by the event with <name>=BER_CHANGE and <filter-type>=9 x 10-4, thus expressing a change of the pre-FEC BER above the threshold. In case the pre-FEC

BER exceeds 9 x 10-4 due to a soft failure, the state machine evolves to the "Fec-Baud-Adapt" state and an adaptation to a more robust FEC of 20% (executed by the attribute <execute>) is performed. The system can return to the "Steady" state if the pre-FEC BER goes below another pre-defined threshold and the FEC is reconfigured to 7%.

7. Appendix

This appendix reports the YANG models code and the related tree.

7.1. YANG model for FSM - Tree

```
module: finite-state-machine
   +--rw current-state?   leafref
   +--rw states
      +--rw state [id]
         +--rw id             state-id-type
         +--rw description?   string
         +--rw transitions
            +--rw transition [name type]
               +--rw name           string
               +--rw type           transition-type
               +--rw description?   string
               +--rw filters
               |  +--rw filter [filter-id]
               |     +--rw filter-id    yp:filter-id
               +--rw actions
                  +--rw action [id]
                     +--rw id             transition-id-type
                     +--rw type           enumeration
                     +--rw conditional
                     |  +--rw statement    string
                     |  +--rw true
                     |  |  +--rw execute
                     |  |  +--rw next-action?   transition-id-type
                     |  |  +--rw next-state?    leafref
                     |  +--rw false
                     |     +--rw execute
                     |     +--rw next-action?   transition-id-type
                     |     +--rw next-state?    leafref
                     +--rw simple
                        +--rw execute
                        +--rw next-action?   transition-id-type
                        +--rw next-state?    leafref
```

7.2. YANG model for FSM - Code

```
 module transitions {

   namespace "http://sssup.it/transitions";

   prefix ev;


   import ietf-yang-push {

     prefix yp;

   }


   organization

     "Scuola Superiore Sant'Anna Network and Services Laboratory";


   contact

     " Editor: Matteo Dallaglio

               <mailto:m.dallaglio@sssup.it>

     ";


   description

     "This module contains a YANG definitions of events and generic
     reactions.";


   revision 2016-03-15 {

     description "Initial Revision.";

     reference

       "RFC xxxx: A YANG data model for the description of events and
```

```
        reactions";
    }

    // identity statements

    identity TRANSITION {
        description "Base for all types of event";
    }

    identity ON_CHANGE {
        base TRANSITION;
        description
          "The event when the database changes.";
    }

    // typedef statements

    typedef transition-type {
      type identityref {
        base TRANSITION;
      }
    }

    typedef transition-id-type {
```

```
    type uint32;
  }



  // grouping statements
  grouping action-block {
    leaf id {
      type transition-id-type;
    }
    leaf type {
      type enumeration {
        enum CONDITIONAL_OP;
        enum SIMPLE_OP;
      }
      mandatory true;
    }

    grouping execution-top {
      anyxml execute {
        description "Represent the action to perform";
      }
      leaf next-action {
        type transition-id-type;
        description "the id of the next action to execute";
```

```
      }
    }

    container conditional {
      when "../type = 'CONDITIONAL_OP'";
      leaf statement {
        type string;
        mandatory true;
        description
          "The statement to be evaluated before execution.
          E.g. if a=b";
      }
      container true {
        uses execution-top;
      }
      container false {
        uses execution-top;
      }
    }

    container simple {
      when "../type = 'SIMPLE_OP'";
      description
        "Simple execution of an action without checking any condition";
```

```
      uses execution-top;
    }
  }

  grouping action-top {
    list action {
      key "id";
      ordered-by user;
      uses action-block;
    }
  }

  grouping on-change {
    description
      "Event occuring when a modification of one or more
       objects occurs";

    container filters {
      description
        "This container contains a list of configurable filters
         that can be applied to subscriptions.  This facilitates
         the reuse of complex filters once defined.";
      list filter {
        key "filter-id";
```

```
         description
           "A list of configurable filters that can be applied to
            subscriptions.";
         leaf filter-id {
           type yp:filter-id;
           description
             "An identifier to differentiate between filters.";
         }
         uses yp:datatree-filter;
       }
     }
   }

   grouping transition-top {
     leaf name {
       type string;
       mandatory true;
     }

     leaf type {
       type transition-type;
       mandatory true;
     }
```

```
    leaf description {
      type string;
    }

    // list of all possible events
    uses on-change {
      when "type = 'ON_CHANGE'";
    }

    container actions {
      uses action-top;
    }
  }

  grouping transitions-top {
    container transitions {
      list transition {
        key "name type";
        uses transition-top;
      }
    }
  }
```

```
   // data definition statements

   uses transitions-top;

   // extension statements

   // feature statements

   // augment statements

   // rpc statements

   // notification statements


 }//module transitions

 module finite-state-machine {
   namespace "http://sssup.it/fsm";
   prefix fsm;

   import transitions {
     prefix ev;
   }
```

```
   organization
     "Scuola Superiore Sant'Anna Network and Services Laboratory";


   contact
     " Editor: Matteo Dallaglio
               <mailto:m.dallaglio@sssup.it>
     ";


   description
     "This module contains a YANG definitions of a generic finite state
     machine.";


   revision 2016-03-15 {
     description "Initial Revision.";
     reference
       "RFC xxxx:";
   }


   // identity statements


   // typedef statements


   typedef state-id-type {
     type uint32;
   }
```

```
   // grouping statements
   grouping state-top {
     leaf id {
       type state-id-type;
     }

     leaf description {
       type string;
     }




     grouping next-state-top {
       leaf next-state {
           type leafref {
             path "../../../../../../../../../../states/state/id";
           }
           description "Id of the next state";
         }
     }

    uses ev:transitions-top {
     augment "transitions/transition/actions/action/conditional/true" {
         uses next-state-top;
```

```
       }

     augment "transitions/transition/actions/action/conditional/false" {
         uses next-state-top;
       }
     augment "transitions/transition/actions/action/simple" {
         //uses next-state-top;
         leaf next-state {
           type leafref {
             path "../../../../../../../../states/state/id";
           }
           description "Id of the next state";
         }
       }
    }

   }



   grouping states-top {
     leaf current-state {
       type leafref {
         path "../states/state/id";
```

```
        }
      }

      container states {
        list state {
          key "id";
          uses state-top;
        }
      }
    }

    // data definition statements

    uses states-top;

    // extension statements

    // feature statements

    // augment statements.
```

```
   // rpc statements


   // notification statements




 }//module fsm
```

7.3. Example of values for the YANG model

FIELD NAME	YANG DATA TYPE	VALUE
Current State	leafref	"an existing state id in the FSM"
State		
id	uint32	1
name	string	Steady
description	string	"whatever string"
transition		
name	string	"whatever string"
type	enum	BER_CHANGE
description	string	"whatever string"
filter		
filter-id	uint32	2
filter-type	anyxml or xpath	BER>0.0009
action		
id	uint32	3
type	enum	SIMPLE
statement	string	"whatever string"
execute	anyxml	"this recalls an RPC where the FEC value is expressed"
next-operation	uint32	NULL
next-state	leafref	"an existing state id in the FSM"

8. Acknowledgements

This work has been partially supported by the European Commission through the H2020 ORCHESTRA (Optical peRformanCe monitoring enabling dynamic networks using a Holistic cross-layEr, Self-configurable Truly flexible approach, grant agreement no: H2020-645360) project. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

9. Other Contributors

Matteo Dallaglio (Scuola Superiore Sant'Anna), Andrea Di Giglio (Telecom Italia), Giacomo Bernini (Nextworks).

10. Security Considerations

TBD

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>.

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

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

[RFC7491] King, D. and A. Farrel, "A PCE-Based Architecture for Application-Based Network Operations", RFC 7491, DOI 10.17487/RFC7491, March 2015, <https://www.rfc-editor.org/info/rfc7491>.

11.2. Informative References

[I-D.brockners-inband-oam-requirements]
Brockners, F., Bhandari, S., Dara, S., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mozes, D., Mizrahi, T., <>, P., and r. remy@barefootnetworks.com, "Requirements for In-situ OAM", draft-brockners-inband-oam-requirements-03 (work in progress), March 2017.

[I-D.fioccola-ippm-multipoint-alt-mark]
Fioccola, G., Cociglio, M., Sapio, A., and R. Sisto, "Multipoint Alternate Marking method for passive and hybrid performance monitoring", draft-fioccola-ippm-multipoint-alt-mark-01 (work in progress), October 2017.

[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-17 (work in progress), October 2017.

[I-D.ietf-ippm-alt-mark]
Fioccola, G., Capello, A., Cociglio, M., Castaldelli, L., Chen, M., Zheng, L., Mirsky, G., and T. Mizrahi, "Alternate Marking method for passive and hybrid performance monitoring", draft-ietf-ippm-alt-mark-13 (work in progress), October 2017.

[I-D.song-opsawg-dnp4iq]
Song, H. and J. Gong, "Requirements for Interactive Query with Dynamic Network Probes", draft-song-opsawg-dnp4iq-01 (work in progress), June 2017.

[I-D.vergara-ccamp-flexigrid-yang]
Madrid, U., Perdices, D., Lopezalvarez, V., Dios, O., King, D., Lee, Y., and G. Galimberti, "YANG data model for Flexi-Grid Optical Networks", draft-vergara-ccamp-flexigrid-yang-05 (work in progress), July 2017.

[I-D.zhang-ccamp-l1-topo-yang]
zhenghaomian@huawei.com, z., Fan, Z., Sharma, A., and X. Liu, "A YANG Data Model for Optical Transport Network Topology", draft-zhang-ccamp-l1-topo-yang-07 (work in progress), April 2017.

Authors' Addresses

Nicola Sambo
Scuola Superiore Sant'Anna
Via Moruzzi 1
Pisa 56124
Italy

Email: nicola.sambo@sssup.it

Piero Castoldi
Scuola Superiore Sant'Anna
Via Moruzzi 1
Pisa 56124
Italy

Email: piero.castoldi@sssup.it

Giuseppe Fioccola
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: giuseppe.fioccola@telecomitalia.it

Filippo Cugini
CNIT
Via Moruzzi 1
Pisa 56124
Italy

Email: filippo.cugini@cnit.it

Haoyu Song
Huawei
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: haoyu.song@huawei.com

Tianran Zhou
Huawei
156 Beiqing Road
Beijing 100095
China

Email: zhoutianran@huawei.com

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 22, 2019

N. Sambo
P. Castoldi
Scuola Superiore Sant'Anna
G. Fioccola
Huawei Technologies
F. Cugini
CNIT
H. Song
T. Zhou
Huawei

May 21, 2019

YANG model for finite state machine
draft-sambo-netmod-yang-fsm-05

Abstract

Network operators and service providers are facing the challenge of deploying systems from different vendors while looking for a trade-off among transmission performance, network device reuse, and capital expenditure without the need of being tied to single vendor equipment. The deployment and operation of more dynamic and programmable network infrastructures can be driven by adopting model-driven and software-defined control and management paradigms. In this context, YANG enables to compile a set of consistent vendor-neutral data models for networks and components based on actual operational needs emerging from heterogeneous use cases. This document proposes YANG models to describe events, operations, and finite state machine of YANG-defined network elements. The proposed models can be applied in several use cases: i) in the context of optical networks to pre-instruct data plane devices (e.g., an optical transponder) on the actions to be performed (e.g., code adaptation) in case some events, such as physical layer degradations, occur; ii) in general data networks, network telemetry applications can define and embed custom data probes into data plane devices. A probe in many cases can be modeled as an FSM; iii) the monitoring of packet loss and delay through a network clustering approach; iv) for re-routing in optical networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Networks are evolving toward more programmability, flexibility, and multi-vendor interoperability. Multi-vendor interoperability can be applied in the context of nodes, i.e. a node composed of components provided by different vendors (named fully disaggregated white box) is assembled under the same control system. This way, operators can optimize costs and network performance without the need of being tied to single vendor equipment. NETCONF protocol RFC6241 [RFC6241] based on YANG data modeling language RFC6020 [RFC6020] is emerging as a candidate Software Defined Networking (SDN) enabled protocol. First, NETCONF supports both control and management functionalities, thus permits high programmability. Then, YANG enables data modeling in a vendor-neutral way. Some recent works have provided YANG models to describe attributes of links (e.g., identification), nodes (e.g., connectivity matrix), media channels, and transponders (e.g., supported forward error correction - FEC) of networks ([I-D.ietf-i2rs-yang-network-topo] [I-D.vergara-ccamp-flexigrid-yang] [I-D.zhang-ccamp-l1-topo-yang]), also including optical technologies. This document presents YANG models to describe events, operations, and finite state machine of YANG-defined network elements. Such models can be applied to several use cases. In the context of elastic optical networks (EONs), the model enables a centralized remote network controller (managed by a network operator) to instruct a transponder controller about the actions to perform when certain events (e.g., failures) occur. The actions to be taken and the events can be re-programmed on the device. In general data networks, programmable network telemetry is considered a killer SDN application which can help applications gain unprecedented visibility to network data plane. Instead of providing raw data, network devices can be configured to filter and process data directly on the data plane and only hand preprocessed data to the collector, in order to save data bandwidth and reduce reaction delay ([I-D.song-opsawg-dnp4iq]) . Such configurations can be programed as custom probes and dynamically deployed into data plane devices. A probe in many cases can be modeled as an FSM. Another use case is the monitoring of packet loss and delay through a network clustering approach: in this case, each FSM state is determined by a specific subdivision of the network in Clusters ([I-D.ietf-ippm-multipoint-alt-mark]). Finally, a use case is related to enable automatic local reconfiguration of a backup route in optical networks.

2. Conventions used in this document

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

3. Terminology

ABNO: Application-Based Network Operations

BER: Bit Error Rate

EON: Elastic Optical Network

FEC: Forward Error Correction

FSM: Finite State Machine

NETCONF: Network Configuration Protocol

OAM: Operation Administration and Maintenance

SDN: Software Defined Network

YANG: Yet Another Network Generator

DNP: Dynamic Network Probe

AMM: Alternate Marking Method

4. Example of application

4.1. Pre-programming resiliency schemes in EONs

EONs (optical networks based on flexible grid supporting circuits of different bandwidth) are expected to employ flexible transponders, i.e. transponders supporting multiple bit rates, multiple modulation formats, and multiple codes. Such transponders permits the (re-) configuration of the bit rate value based on traffic requirements, as well as the configuration of the modulation format and code based on the physical characteristics of a path (e.g., quadrature phase shift keying is more robust than 16 quadrature amplitude modulation). This way, transmission parameters can be (re-) configured based on physical layer changes. The YANG model presented in this draft enables to pre-program reconfiguration settings of data plane devices in case of failures or physical layer degradations. In particular, soft failures are assumed. Soft failures imply transmission performance degradation, in turns a bit error rate (BER) increase, e.g. due to the ageing of some network devices. Without loosing generality, the ABNO architecture is assumed for the control and management of EONs (RFC7491 [RFC7491]). Considering the state of the art, when pre-FEC BER passes above a predefined threshold, it is expected that an alarm is sent to the OAM Handler, which communicates with the ABNO controller that may trigger an SDN controller (that could be the Provisioning Manager of ABNO RFC7491 [RFC7491]) for computing new transmission parameters. The involved ABNO modules are shown in the simplified ABNO architecture of Fig. 1. Then, transponders are reconfigured. When alarms related to several connections impacted by the soft failure are generated, this procedure may be particularly time consuming. The related workflow for transponder reconfiguration is shown in Fig. 2. The proposed model enables an SDN controller to instruct the transponder about reconfiguration of new transmission parameters values if a soft failure occurs. This can be done before the failure occurs (e.g., during the connection instantiation phase or during the connection service), so that data plane devices can promptly reconfigure themselves without querying the SDN controller to trigger an on-demand recovery. This is expected to speed up the recovery process from soft failures. The related flow chart is shown in Fig. 3. The whole mechanism is based on a finite state machine where each state is associated to a specific configuration of transmission parameters (e.g., modulation format). The transition from a state to another state is triggered by specific events at the physical layer such as the bit error rate above a threshold. The transition from a state to another state implies a set of actions, including the change of transmission parameters (e.g., modulation format), which are actually suitable for the current condition at the physical layer. Moreover, since transmission and receiver must be synchronized about the transmission settings (modulation format and so no) for a proper transmission, another action consists of this synchronization. Thus, when the transponder at the receiver side decides to change its transmission parameters based on the monitored BER, the remote transponder at the transmitter side has to do the same state transition. In particular, the transponder at the receiver side sends a message to the transmitter to synchronize about the transmission parameters to be adopted. This message can be sent over a control channel. This way both the transmitter and receiver operates with the same transmission parameters: e.g. the format, FEC, and so on. No central controller is involved at this stage, only a notification can be sent to the central controller to inform it about the successful reconfiguration.

```
           ____________            ____________
          |    ABNO    |          |    OAM     |
          | controller |  ------  |   Handler  |
          |____________|          |____________|

             |                          |
             |                          |
             |                          |
          ____________                  |
         |    SDN     |                 |
         | controller |                 |
         |____________|                 |
                                        |
              |                         |
              |                         |
              |                         |
           ______________________________
          |           Client             |
          |           network            |
          |______________________________|
```

Figure 1: Assumed ABNO functional modules

```
                  _____________________
                 |         1           |
                 |Sending alarm to the |
                 |     OAM Handler     |
                 |                     |
                 |_____________________|
                           |
                           |
                           |
                 _____________________
                |         2           |
                |      Trigger        |
                |   SDN Controller    |
                |                     |
                |_____________________|
                           |
                           |
                           |
                 _____________________
                |         3           |
                |   Computation of    |
                |  new transmission   |
                |    parameters       |
                |_____________________|
                          |
                          |
                          |
                 _____________________
                |        4            |
                |    Data plane       |
                |    reconfiguration  |
                |                     |
                |_____________________|
```

Figure 2: Flow chart of the expected state-of-the-art approach

```
           _______________________
          |          1            |
          | Instructing the local |
          |     controller of     |
          |  data plane devices   |
          |_______________________|
                     |
                     |
                     |
           _______________________
          |          2            |
          | Local reconfiguration |
          |     upon failure      |
          |       detection       |
          |_______________________|
                     |
                     |
                     |
           _______________________
          |          3            |
          |                       |
          |     notification      |
          |                       |
          |_______________________|
```

Figure 3: Flow chart of the approach exploiting YANG models in this draft

4.2. Deploying Dynamic Probes for Programmable Network Telemetry

In the past, network data analytics was considered a separate function from networks. They consume raw data extracted from networks through piecemeal protocols and interfaces. With the advent of user programmable data plane, we expect a paradigm shift that makes the data plane be an active component of the data telemetry and analytics solution. The programmable in-network data preprocessing is efficient and flexible to offload some light-weight data processing through dynamic data plane programming or configuration. A universal network data analytics platform built on top of this enables a tight and agile network control and OAM feedback loop. A proposed dynamic network telemetry system architecture is illustrated in Fig.4.

An application translates its data requirements into a set of Dynamic Network Probes (DNP) targeting a subset of data plane devices. After the probes are deployed, each probe conducts its corresponding in-network data preprocessing and feeds the preprocessed data to the collector. The collector finishes the data post-processing and presents the results to the data-requesting application.

```
+--------------------------------------+
|  network telemetry applications      |
+--------------------------------------+
       ^                   |
       |                   V
       |          +---------------------+
       |          | DNP compile/config  |
       |          +---------------------+
       |                   |
       |                   V
+----------------+ +---------------------+
|data collection| | Probe deployment    |
+----------------+ +---------------------+
    ^   ^   ^            |   |   |
    |   |   |            V   V   V
+---------------------------------------+
|  network data plane devices           |
| (in-network data preprocessing)       |
+---------------------------------------+
```

Figure 4: Deploy dynamic network probes using YANG FSM models

Many DNPs can be modeled as FSM which are configured to capture specific events. Here FSMs essentially preprocess the raw stream data and only report the necessary data to subscribing applications.

For example, a congestion control application needs to monitor the router buffer occupancy. Instead of polling the buffer depth periodically, it is only interested in the real-time events when the buffer depth crosses a low and a high threshold. We can install a probe to achieve this data plane function and the probe can be modeled as a three-state FSM. Each state represents a buffer region: below the low threshold, above the high threshold, and in between the two thresholds. A possible state transition is checked against the buffer depth for each incoming and outgoing packet. Whenever a state transition happens, an event is generated and reported to the application. This approach significantly reduces the amount of data sent to the application and also allows a timely event notification.

For another example, an application would like to monitor the delay experienced by a flow. The packet delay on its forwarding path can be acquired by using iOAM [I-D.brockners-inband-oam-requirements]. However, the application only needs to know that N consecutive flow packets experience a delay longer than T. Instead of forwarding the raw delay data to the application, a probe can be deployed to detect the event. Similarly, the probe can be modeled as an FSM.

4.3. IP Performance Measurements on multipoint-to-multipoint large Networks

Networks offer rich sets of network performance measurement data, but traditional approaches run into limitations. One reason for this is the fact that in many cases, the bottleneck is the generation and export of the data and the amount of data that can be reasonably collected from the network runs into bandwidth and processing constraints in the network itself. In addition, management tasks related to determining and configuring which data to generate lead to significant deployment challenges.

In order to address these issues, an SDN controller application orchestrates network performance measurements tasks across the network to allow an optimized monitoring. In fact the IP Performance Measurement SDN Controller Application in Figure 5 can calibrate how deep can be obtained monitoring data from the network by configuring measurement points roughly or meticulously. This can be established by using the feedback mechanism reported in Figure 5.

For instance, the SDN Controller can configure initially an end to end monitoring between ingress points and egress points of the network. If the network does not experiment issues, this approximate monitoring is good enough and is very cheap in terms of network resources. But, in case of problems, the SDN Controller becomes aware of the issues from this approximate monitoring and, in order to localize the portion of the network that has issues, configures the measurement points more exhaustively. So a new detailed monitoring is performed. After the detection and resolution of the problem the initial approximate monitoring can be used again. This idea is general and can be applied to different performance measurements techniques both active and passive (and hybrid).

```
+---------------------------------------+
|      IP Performance Measurement       |
|      SDN Controller Application       |
+---------------------------------------+
    ^   ^   ^            |   |   |
    |   |   |            v   v   v
+---------------------------------------+
|          Multipoint Network           |
+---------------------------------------+
```

Figure 5: Feedback mechanism on multipoint-to-multipoint large Networks

One of the most efficient methodology to perform packet, loss delay and jitter measurements both in an IP and Overlay Networks is the Alternate Marking method, as presented in RFC8321 [RFC8321] and [I-D.ietf-ippm-multipoint-alt-mark].

This technique can be applied to point-to-point flows but also to multipoint.to-multipoint flows (see RFC8321 [RFC8321] and [I-D.ietf-ippm-multipoint-alt-mark]). The Alternate Marking method creates batches of packets by alternating the value of 1 or 2 bits of the packet header. These batches of packets are unambiguously recognized over the network and the comparison of packet counters permits the packet loss calculation. The same idea can be applied for delay measurement by selecting special packets with a marking bit dedicated for delay measurements. This method needs two counters each marking period for each flow under monitor. For this reason by considering n measurement points and n monitored flows, the order of magnitude of the packet counters for each time interval is n*n*2 (1 per color).

Multipoint Alternate Marking, described in [I-D.ietf-ippm-multipoint-alt-mark], aims to reduce this value and makes the performance monitoring more flexible in case a detailed analysis is not needed.

It is possible to monitor a Multipoint Network without examining in depth by using the Network Clustering (subnetworks that are portions of the entire network that preserve the same property of the entire network). So in case there is packet loss or the delay is too high the filtering criteria could be specified more in order to perform a per flow detailed analysis, as described in RFC8321 [RFC8321].

An application of the multipoint performance monitoring can be done by using FSM (each state is a composition of clusters) and feedback mechanism where the SDN Controller is the brain of the network and can manage flow control to the switches and routers and, in the same way, can calibrate the performance measurements depending on the necessity.

4.4. Re-routing in optical networks

FSM can be also applied for rerouting purposes as dynamic restoration in optical networks. In such a use case all the nodes along the backup route of a media channel should hold a FSM indicating the reconfigurations to be performed in case that media channel is rerouted along the backup path. Note that resources along the backup route are not reserved neither configured during a normal operation of the network as instead it happens for protection.

The proposed approach aims at achieving a sort of hybrid centralized/ distributed rerouting solution applicable to all the networks controlled with NETCONF. More specifically, the decision of the backup route is taken centrally by the SDN controller, but it is acted in a distributed way through FSM when a problem appears.

In particular, the transmitter, the receiver, and the ROADMs along the backup route have installed in their agent a FSM including a "re-routing" state, associated to a specific media channel. For the transmitter and receiver, this state is associated to the transmission parameters of the backup media channel: e.g., modulation format, central frequency, FEC, and so on. Regarding the ROADMs, the "re-routing" state is associated to the cross connections of the backup route.

When a link failure occurs (e.g., fiber cut), the receiver reveals a loss of light, which triggers the transition to the "re-routing" state. Thus, the receiver can set itself to the new transmission parameters. Moreover, similarly to the message sent over a control channel to the transmitter in Sec. 4.1, the receiver sends a message to the transmitter and the backup Reconfigurable Add-Drop Multiplexer (ROADMs). The task of this message is to simply trigger a state transition to the "re-routing" state so that each backup ROADM and the transmitter can apply the proper reconfigurations. This way, the SDN controller is not interrogated during the failure and the recovery can be speeded up. After reconfigurations, the SDN controller can be notified and the SDN controller can tear down the primary path. Backup routes can be reprogrammed based on the network states evolutions (e.g., link and spectrum occupancy).

5. YANG for finite state machine (FSM)

This model defines a list of states and transitions to describe a generic finite state machine (FSM). The related code and tree are shown in the Appendix.

```
 <current-state>: it defines the current state of the FSM.
 <states>: this element defines the FSM as follows.
           <state>: this list defines all the FSM states.
                  <id>: this leaf attribute of <state> defines the
                  identifier of the state
                  <name>: this leaf attribute of <state> defines the
                  name of the state
                  <description>: this leaf is a "string" describing the
                  state
                  <transitions>: this attribute defines a list of
                  transitions to other states in the FSM.
                          <name>: this attribute defines the name of a
```

transition
<type>: this attribute defines the type of the transition from a pool of possible transition types predefined inside the YANG model. Together with the <name> attribute, it uniquely identifies the transition.
<description>: this optional attribute is a "string" describing the transition
<filters>: this leaf is a list of input parameters related to the transition. This attribute enables to further express a transition: as an example, if a transition can be triggered by a parameter (e.g., a monitored performance parameter) exceeding a threshold (as in Sec. 5), an element of the list defines this threshold. Thus, if the parameter is outside the threshold, the transition is taken, otherwise not.

<filter>: this leaf of <filters> defines a filter parameter.
<filter-id>: this leaf of <filters> define the identifier number associated with the <filter> attribute.

<actions>: this attribute defines a list of actions to take during the transition.

<action>: this attribute is the list of actions

<id>: this leaf of <action> defines the identifier number of an action.
<type>: this leaf of <action> defines the type of an action.
<simple>: this leaf defines (differently from <conditional> detailed below) an action that has to be directly executed.

<execute>: this attribute recalls an RPC encapsulating the effective task (action) to be executed by the hardware. If more actions (e.g., "A" and "B"), defined in the <action> list, have to be executed, these actions can be executed sequentially according to the <next-action> attribute detailed below. Thus, by

referring to the tree of the Appendix, when an action ("A") is executed, the <next-action> attribute will bring to another action ("B"). If more actions have to be executed in parallel (e.g., "A" & "B"), not sequentially, an element of the <action> list should be defined to express an action (e.g., "A&B") consisting of more actions to be executed in parallel.

<next-action>: this attribute defines the identification number of a next action that has to be taken. The <next-action> can assume a NULL value.

<conditional>: this leaf enables a check ("true" or "false") to be verified before executing the action. Based on the check, the proper attributes <execute> and <next-operation> are considered.

<statement>: this leaf of <conditional> defines the condition to be verified before executing the action.

<true>: this leaf of <conditional> defines a result of the check associated to <statement>. Proper <execute> and <next-operation> attributes are associated with this result of the check.

<false>: this leaf of <conditional> defines a result of the check associated to <statement>. Proper <execute> and <next-operation>

attributes are associated with this result of the check.

<next-state>: this attribute defines the next state of FSM when an action is executed.

6. Implementation of the pre-programming resiliency schemes in EONs

These presented model can be used to enable a centralized network controller, managed by a network operator, to instruct data plane hardware on its reconfiguration if some events, such as a failure or physical layer degradation, occur. As an example, an optical signal impacted by a soft failure (i.e., a physical layer degradation inducing a pre forward error correction bit error rate increase - pre-FEC) can be maintained by adapting the FEC of the signal itself. This action to be taken and, more in general operations to be executed depending on critical events, can be (re-) programmed on the transponder by (re-) sending a NETCONF <edit-config> message to the device controller including a FSM defined by the YANG model. Such a system has the main goal to speed up the reaction of the network to certain events/faults and to alleviate the workload of the centralized controller. The speed up derives from the fact that the centralized controller is able to pre-compute and pre-configure on the network devices the actions to take when an event occurs taking into account a global view and knowledge of the network. In this way, the device is already aware of the actions to be locally applied to reconfigure a connection, avoiding to inform the controller and to wait for the response indicating what to do. Consequently, part of the workload is also removed from the centralized controller. When the reaction is successfully completed in the data plane, the centralized controller can be notified about the faults and the taken action. A flexible transponder supporting two FEC types, 7% and 20%, is considered. A two-states FSM is also assumed. The states have <name> attribute set to "Steady" and "Fec-Baud-Adapt", respectively. In the "Steady" state, the signal is in a healthy condition, adopting a 7% FEC, with a pre-FEC BER below an assigned threshold of 9 x 10-4. A transition from this state can be triggered by the event with <name>=BER_CHANGE and <filter-type>=9 x 10-4, thus expressing a change of the pre-FEC BER above the threshold. In case the pre-FEC BER exceeds 9 x 10-4 due to a soft failure, the state machine evolves to the "Fec-Baud-Adapt" state and an adaptation to a more robust FEC of 20% (executed by the attribute <execute>) is performed. The system can return to the "Steady" state if the pre-FEC BER goes below another pre-defined threshold and the FEC is reconfigured to 7%.

7. Appendix

This appendix reports the YANG models code and the related tree.

7.1. YANG model for FSM - Tree

```
module: ietf-fsm
   +--rw current-state?   leafref
   +--rw states
      +--rw state [id]
         +--rw id             state-id-type
         +--rw description?   string
         +--rw transitions
            +--rw transition [name type]
               +--rw name           string
               +--rw type           transition-type
               +--rw description?   string
               +--rw filters
               |  +--rw filter [filter-id]
               |     +--rw filter-id    uint32
               +--rw actions
                  +--rw action [id]
                     +--rw id             transition-id-type
                     +--rw type           enumeration
                     +--rw conditional
                     |  +--rw statement    string
                     |  +--rw true
                     |  |  +--rw execute
                     |  |  +--rw next-action?   transition-id-type
                     |  |  +--rw next-state?    leafref
                     |  +--rw false
                     |     +--rw execute
                     |     +--rw next-action?   transition-id-type
                     |     +--rw next-state?    leafref
                     +--rw simple
                        +--rw execute
                        +--rw next-action?   transition-id-type
                        +--rw next-state?    leafref
```

7.2. YANG model for FSM - Code

<CODE BEGINS> file "ietf-fsm@2016-03-15.yang"

```
 module ietf-fsm {

   namespace "http://sssup.it/fsm";
```

```
   prefix fsm;




   identity TRANSITION {

       description "Base for all types of event";

   }


   identity ON_CHANGE {

       base TRANSITION;

       description

         "The event when the database changes.";

   }


   // typedef statements


   typedef transition-type {

        description "it defines the type of transition (event)";

     type identityref {

       base TRANSITION;

     }

   }


   typedef transition-id-type {
        description "it defines the id of the transition (event)";
```

```
     type uint32;

   }




   // grouping statements

   grouping action-block {

        description "it defines the action to perform when a transition
        occurs";

     leaf id {

description "it refers to the id of the transition";
       type transition-id-type;

     }

     leaf type {

description "it defines if the action has to be simply executed or if
a conditional statement has to be checked before execution";

       type enumeration {

         enum "CONDITIONAL_OP" {
description "it defines the type CONDITIONAL OPERATION to check a
statement before execution";
}

         enum "SIMPLE_OP" {
description "it defines the type SIMPLE OPERATION: i.e., an operation
to be directly executed;
                }

       }

       mandatory true;

     }



     grouping execution-top {
```

```
         description "it defines the execution attribute";

        anyxml execute {

          description "Represent the action to perform";

        }

        leaf next-action {

          type transition-id-type;

          description "the id of the next action to execute";


        }

      }


      container conditional {

         description "it defines the container CONDITIONAL";

        when "../type = 'CONDITIONAL_OP'";

        leaf statement {

          type string;

          mandatory true;

          description

            "The statement to be evaluated before execution.

            E.g. if a=b";

        }

        container true {

description "it is referred to the result TRUE of a conditional
statement";

          uses execution-top;
```

```
       }

       container false {

description "it is referred to the result FALSE of a conditional
statement ";

         uses execution-top;

       }

     }



     container simple {
description "Simple execution of an action without checking any
condition";

       when "../type = 'SIMPLE_OP'";


       uses execution-top;

     }

   }


   grouping action-top {


description "it defines the grouping of action";

     list action {


description "it defines the list of actions";

       key "id";

       ordered-by user;

       uses action-block;

     }
```

```
   }

   grouping on-change {
     description
       "Event occuring when a modification of one or more
        objects occurs";

     container filters {
       description
         "This container contains a list of configurable filters
          that can be applied to subscriptions.  This facilitates
          the reuse of complex filters once defined.";
       list filter {
         key "filter-id";

         description
           "A list of configurable filters that can be applied to
            subscriptions.";
         leaf filter-id {
           type uint32;
           description
             "An identifier to differentiate between filters.";
         }
       }
     }
```

```
   }


   grouping transition-top {
description "it defines the grouping transition";
     leaf name {
description "it defines the transition name";
       type string;
       mandatory true;
     }


     leaf type {

description "it defines the transition type";
       type transition-type;
       mandatory true;
     }


     leaf description {

description "it describes the transition ";
       type string;
     }


     // list of all possible events
     uses on-change {
```

```
       when "type = 'ON_CHANGE'";
     }

     container actions {
description "it defines the container action";
       uses action-top;
     }
   }

   grouping transitions-top {
description "it defines the grouping transition";
     container transitions {
description "it defines the container transitions";
       list transition {
description "it defines the list of transitions";
         key "name type";
         uses transition-top;
       }
     }
   }

   // data definition statements
```

```
   uses transitions-top;

   // extension statements

   // feature statements

   // augment statements


   organization
     "Scuola Superiore Sant'Anna Network and Services Laboratory";

   contact
     " Editor: Matteo Dallaglio
               <mailto:m.dallaglio@sssup.it>
     ";

   description
     "This module contains a YANG definitions of a generic finite state
     machine.";

   revision 2016-03-15 {
     description "Initial Revision.";
     reference
       "RFC xxxx:";
```

```
   }

   // identity statements

   // typedef statements

   typedef state-id-type {
description "it defines the id type of the states";
     type uint32;
   }

   // grouping statements
   grouping state-top {
description "it defines the grouping state";
     leaf id {
description "it defines the id of a transition";
       type state-id-type;
     }

     leaf description {
description "it describes a transition";

       type string;
```

```
     }




     grouping next-state-top {

description "it defines the grouping for the next state";

       leaf next-state {

description "it defines the next state";
           type leafref {

description "it refers to its id";
             path "../../../../../../../../../states/state/id";
           }
           description "Id of the next state";
         }
     }


    uses transitions-top {
     augment "transitions/transition/actions/action/conditional/true" {
         uses next-state-top;

       }


     augment "transitions/transition/actions/action/conditional/false" {
```

```
         uses next-state-top;

       }

     augment "transitions/transition/actions/action/simple" {

         //uses next-state-top;

         leaf next-state {

description "it defines the next state";


           type leafref {

 description "it refers to its id";
             path "../../../../../../../../states/state/id";

           }

           description "Id of the next state";

         }

       }

    }


   }




   grouping states-top {

description "it defines the grouping states";

     leaf current-state {

description "it defines the current state";

       type leafref {
```

```
description "it refers to its id";
         path "../states/state/id";


       }

     }



     container states {
description "it defines the container states";

       list state {

   description "it defines the list of states";

         key "id";

         uses state-top;

       }

     }

   }




   // data definition statements





   uses states-top;



   // extension statements



   // feature statements
```

```
   // augment statements.


   // rpc statements


 }//module fsm
```

<CODE ENDS>

7.3. Example of values for the YANG model

FIELD NAME	YANG DATA TYPE	VALUE
Current State	leafref	"an existing state id in the FSM"
State		
id	uint32	1
name	string	Steady
description	string	"whatever string"
transition		
name	string	"whatever string"
type	enum	BER_CHANGE
description	string	"whatever string"
filter		
filter-id	uint32	2
filter-type	anyxml or xpath	BER>0.0009
action		
id	uint32	3
type	enum	SIMPLE
statement	string	"whatever string"
execute	anyxml	"this recalls an RPC where the FEC value is expressed"
next-operation	uint32	NULL
next-state	leafref	"an existing state id in the FSM"

8. Acknowledgements

This work has been partially supported by the European Commission through the H2020 ORCHESTRA (Optical peRformanCe monitoring enabling dynamic networks using a Holistic cross-layEr, Self-configurable Truly flexible approach, grant agreement no: H2020-645360) project. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

9. Other Contributors

Matteo Dallaglio (Scuola Superiore Sant'Anna), Andrea Di Giglio (Telecom Italia), Giacomo Bernini (Nextworks).

10. Security Considerations

TBD

11. IANA Considerations

TBD

12. References

12.1. Normative References

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

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

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

[RFC7491] King, D. and A. Farrel, "A PCE-Based Architecture for Application-Based Network Operations", RFC 7491, DOI 10.17487/RFC7491, March 2015, <https://www.rfc-editor.org/info/rfc7491>.

12.2. Informative References

[I-D.brockners-inband-oam-requirements]
Brockners, F., Bhandari, S., Dara, S., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mozes, D., Mizrahi, T., Lapukhov, P., and r. remy@barefootnetworks.com, "Requirements for In-situ OAM", draft-brockners-inband-oam-requirements-03 (work in progress), March 2017.

[I-D.ietf-i2rs-yang-network-topo]
Clemm, A., Medved, J., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A Data Model for Network Topologies", draft-ietf-i2rs-yang-network-topo-20 (work in progress), December 2017.

[I-D.ietf-ippm-multipoint-alt-mark]
Fioccola, G., Cociglio, M., Sapio, A., and R. Sisto, "Multipoint Alternate Marking method for passive and hybrid performance monitoring", draft-ietf-ippm-multipoint-alt-mark-01 (work in progress), March 2019.

[I-D.song-opsawg-dnp4iq]
Song, H. and J. Gong, "Requirements for Interactive Query with Dynamic Network Probes", draft-song-opsawg-dnp4iq-01 (work in progress), June 2017.

[I-D.vergara-ccamp-flexigrid-yang]
Madrid, U., Perdices, D., Lopezalvarez, V., Dios, O., King, D., Lee, Y., and G. Galimberti, "YANG data model for Flexi-Grid Optical Networks", draft-vergara-ccamp-flexigrid-yang-06 (work in progress), January 2018.

[I-D.zhang-ccamp-l1-topo-yang]
zhenghaomian@huawei.com, z., Fan, Z., Sharma, A., and X. Liu, "A YANG Data Model for Optical Transport Network Topology", draft-zhang-ccamp-l1-topo-yang-07 (work in progress), April 2017.

[RFC8321] Fioccola, G., Ed., Capello, A., Cociglio, M., Castaldelli, L., Chen, M., Zheng, L., Mirsky, G., and T. Mizrahi, "Alternate-Marking Method for Passive and Hybrid Performance Monitoring", RFC 8321, DOI 10.17487/RFC8321, January 2018, <https://www.rfc-editor.org/info/rfc8321>.

Authors' Addresses

Nicola Sambo
Scuola Superiore Sant'Anna
Via Moruzzi 1
Pisa 56124
Italy

Email: nicola.sambo@sssup.it

Piero Castoldi
Scuola Superiore Sant'Anna
Via Moruzzi 1
Pisa 56124
Italy

Email: piero.castoldi@sssup.it

Giuseppe Fioccola
Huawei Technologies
Riesstrasse, 25
Munich 80992
Germany

Email: giuseppe.fioccola@huawei.com

Filippo Cugini
CNIT
Via Moruzzi 1
Pisa 56124
Italy

Email: filippo.cugini@cnit.it

Haoyu Song
Huawei
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: haoyu.song@huawei.com

Tianran Zhou
Huawei
156 Beiqing Road
Beijing 100095
China

Email: zhoutianran@huawei.com