

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

J. Clarke
B. Claise
Cisco Systems, Inc.
July 3, 2017

YANG module for yangcatalog.org
draft-clacla-netmod-model-catalog-00

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

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Status of Work and Open Issues	3
2. Learning from Experience	3
2.1. YANG Module Library	4
2.2. YANG Catalog Data Model	4
2.3. Module Sub-Tree	5
2.4. Compilation Information	6
2.5. Maturity Level	8
2.6. Implementation	8
2.7. Vendor Sub-Tree	9
2.8. Regex Expression Differences	10
3. YANG Catalog Use Cases	10
3.1. YANG Search Metadata	10
3.2. YANG In Images	11
4. YANG Catalog YANG module	11
5. Security Considerations	19
6. IANA Considerations	19
7. References	20
7.1. Normative References	20
7.2. Informative References	20
7.3. URIs	20
Appendix A. Acknowledgments	20
Appendix B. Contributors	21
Authors' Addresses	21

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 <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 contains 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 much more in the future. The industry starts 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 must be kept next to the YANG module specification. The YANG module design 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.

1.1. Status of Work and Open Issues

The top open issues are:

1. Add a leaf to indicate the tree type relative to [I-D.ietf-netmod-revised-datastores]
2. Obtain feedback from vendors and SDOs
3. Socialize module at the IETF and incorporate feedback

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. [TO BE EXPANDED ON THE DIFFERENT OBJECTIVES.] Below are our observations and the justifications for the divergence.

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]
        +--rw name          yang:yang-identifier
        +--rw revision      union
        +--rw datastore*    identityref
        +--rw schema?       inet:uri
        +--rw namespace     inet:uri
        +--rw feature*      yang:yang-identifier
        +--rw deviation* [name revision]
          | +--rw name      yang:yang-identifier
          | +--rw revision  union
          +--rw conformance-type enumeration
        +--rw submodule* [name revision]
          | +--rw name      yang:yang-identifier
          | +--rw revision  union
          | +--rw schema?   inet:uri
          +--rw document-name? string
          +--rw author-email? string
          +--rw compilation-status? enumeration
          +--rw compilation-result? string
          +--rw reference?      string
          +--rw prefix?         string
          +--rw yang-version?   string
          +--rw organization?   string
          +--rw description?    string
```

```

|      +--rw contact?                string
|      +--rw maturity-level?         enumeration
|      +--rw implementations
|      |   +--rw implementation* [vendor platform software-version softwar
e-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 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                enumeration
                    +--rw protocol-version?  string
                    +--rw capabilities*      string
                +--rw modules
                  +--rw module* [name revision]
                    +--rw name                -> ../../../../../../../../../../
../../../../../../../../modules/module/name
                    +--rw revision            -> ../../../../../../../../../../
../../../../../../../../modules/module/revision

```

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. Within the "module" sub-tree, each module is listed using its YANG Module Library [RFC7895] "module-list" grouping information. The yang-catalog module then augments the grouping to add metadata elements that will aid module developers and module consumers alike in understanding the relative maturity, compilation status, and the support contact(s) of each YANG module.

```

+--rw module* [name revision]
  +--rw name                yang:yang-identifier
  +--rw revision             union
  +--rw schema?              inet:uri
  +--rw namespace            inet:uri
  +--rw feature*             yang:yang-identifier
  +--rw deviation* [name revision]
    | +--rw name            yang:yang-identifier
    | +--rw revision        union
  +--rw conformance-type     enumeration
  +--rw submodule* [name revision]
    | +--rw name            yang:yang-identifier
    | +--rw revision        union
    | +--rw schema?         inet:uri
  +--rw document-name?       string
  +--rw author-email?        string
  +--rw compilation-status?  enumeration
  +--rw compilation-result?  string
  +--rw reference?           string
  +--rw prefix?              string
  +--rw yang-version?        string
  +--rw organization?        string
  +--rw description?         string
  +--rw contact?             string
  +--rw module-type?         enumeration
  +--rw maturity-level?      enumeration
  +--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

```

Many of these additional metadata fields are self-explanatory, especially given their descriptions in the module itself. 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 stage of maturity should be taken into account, including YANG modules that are clearly work-in-progress, i.e. that do not validate correctly, either because of a faulty YANG constructs, because of a faulty imported YANG module, or simply because of a warnings. Note that

some of those warnings are not always show-stoppers from a code generation point of view. Nonetheless, the compilation or validation status, along with the compilation output, provide a clear indication on the YANG module development phase and stability.

```

    leaf compilation-status {
      description
        "Status of the module, whether it was possible to compile this YANG mo
dule or
        there are still some errors/warnings.";
      type enumeration {
        enum PASSED {
          value 0;
          description
            "In case that all compilers were able to compile this YANG module
without
            any error/warning.";
        }
        enum PASSED-WITH-WARNINGS {
          value 1;
          description
            "In case that all compilers were able to compile this YANG module
without
            any error, but at least one of them caught some warning.";
        }
        enum FAILED {
          value 2;
          description
            "In case that at least one of compilers found some error while
            compiling this YANG module.";
        }
        enum MISSING {
          value 3;
          description
            "In case that there is not sufficient information about compilatio
n status.";
        }
      }
    }
    leaf compilation-result {
      type string;
      description
        "Result of the compilation explaining specifically what error or warni
ng occurred.
        This is not existing if compilation status is PASSED.";
    }
  }

```

Figure 1: Compilation Status and Compilation Result

The compilation status and result have been added as two extra leaves, for each YANG module.

2.5. Maturity Level

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

```
leaf maturity-level {
  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.";
  type enumeration {
    enum ratified {
      value 0;
      description
        "Maturity of a module that is fully approved (e.g., a standard).";
    }
    enum working-group {
      value 1;
      description
        "Maturity of a module that is actively being developed by a organi
zation towards ratification.";
    }
    enum individual {
      value 2;
      description
        "Maturity of a module that has been initially created, but has no
official
        organization-level status.";
    }
  }
}
```

This enumeration mapping has been implemented for the YANG modules from IETF and BBF. With respect to vendor-specific modules, this same enumeration should be used and mapped to the internal vendor release or development names. Once a module has been completed, fully tested, and is stable, its maturity level should be "ratified".

2.6. 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
```

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.

2.7. 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 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          enumeration
                +--rw protocol-version? string
                +--rw capabilities* string
            +--rw modules
              +--rw module* [name revision]
                +--rw name          -> ../../../../../../../../../../.
./modules/module/name
                                +--rw revision    -> ../../../../../../../../../../.
./modules/module/revision

```

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 ASR9000 routers) instead of only being able to look for YANG modules for a specific platform and software release.

2.8. Regex Expression Differences

Another challenge with 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. YANG In Images

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 images. 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.

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@2017-07-03.yang"
module yang-catalog {
  namespace "urn:ietf:params:xml:ns:yang:yang-catalog";
  prefix yc;

  import ietf-yang-library {
    prefix yanglib;
  }

  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 2017-07-03 {
    description
      "Initial revision.";
```

```
reference "  
  YANG Catalog <https://yangcatalog.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";  
    uses yanglib:module-list;  
  } // end of modules  
  
  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 versi
ons.";
      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 net
work device appliances,
            this will be the operating system version. But for other
YANG module
            implementation, this would be a version of appliance softw
are. Ultimately,
            this should correspond to a version string that will be re
cognizable 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, platform
, software-version.";
            leaf name {
              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 feat
ure set.";
            }
          }
          container protocols {
            description
              "List of the protocols";
            list protocol {
              key name;
              description
                "YANG-based protocol that is used on the device. Thi
s enumeration will
                is expected to be augmented to list other protocol n
ames.";
              leaf name {
                type enumeration {
                  enum netconf {
                    description
                      "NETCONF protocol described in RFC 6241";
                  }
                  enum restconf {
                    description
                      "RESTCONF protocol described in RFC 8040";
                  }
                }
              }
            }
          }
        }
      }
    }
  }

```



```

        description
            "Name of the YANG-based protocol that is supported.
";
    } // end of name
    leaf 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.";
    }
} // end of protocol
} // end of protocols
container modules {
    description
        "Container holding list of modules.";
    list module {
        key "name revision";
        description
            "List of references to YANG modules under specific ven
            dor, platform, software-version,
            software-flavor. Using these references, the complet
            e set of metadata can be
            retrieved for each module.";
        leaf name {
            type leafref {
                path "../.../.../.../.../.../.../.../.../.../.../modules/modu
le/name";
            }
            description
                "Reference to a name of the module that is containe
                d in specific vendor, platform,
                software-version, software-flavor.";
        }
        leaf revision {
            type leafref {
                path "../.../.../.../.../.../.../.../.../.../.../modules/modu
le/revision";
            }
            description
                "Reference to a revision of the module that is cont
                ained in specific vendor,
                platform, software-version, software-flavor.";
        }
    } // end of list module
} // end of container modules
} // end of software-flavor
} // end of software-flavors
} // end of software-version
} // end of software-versions
} // end of platform

```

```

    } // end of platforms
  } // end of vendor
} // end of vendors
} //end of catalog

augment "/catalog/modules/module" {
  uses module-data;
  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 created 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 network dev
ice appliances,
          this will be the operating system version.  But for other YANG mod
ule
          implementation, this would be a version of appliance software.  Ul
timately,
          this should correspond to a version string that will be recognizab
le 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, thi
s could
          be a license, additional software component, or a feature set.";
      }
      leaf os-version {
        type string;
        description
          "Version of the operating system using this module.  This is primar
ily useful if
          the software implementing the module is an application that requir
es a specific
          operating system.";
      }
      leaf feature-set {

```



```

        type string;
        description
            "An optional feature of the software that is required in order to i
implement this
            module. Some form of this must be incorporated in software-versio
n or
            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
useful if
            the software implementing the module is an application that requir
es a
            specific operating system.";
    }
}
}
description
    "This table augments the per-module metadata set and provides details abo
ut
    vendor implementations for each module.";
}

grouping module-data {
    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 string;
        description
            "Contact email of the author who created this module.";
    }
    leaf compilation-status {
        type enumeration {
            enum PASSED {
                value 0;
                description
                    "In case that all compilers were able to compile this YANG module w
ithout
                    any error/warning.";
            }
            enum PASSED-WITH-WARNINGS {
                value 1;
                description
                    "In case that all compilers were able to compile this YANG module w
ithout
                    any error, but at least one of them caught some warning.";
            }
            enum FAILED {
                value 2;

```

```
        description
            "In case that at least one of compilers found some error while
            compiling this YANG module.";
    }
    enum MISSING {
        value 3;
        description
            "In case that there is not sufficient information about compilation
            status.";
    }
    }
    description
        "Status of the module, whether it was possible to compile this YANG mod
        ule or
        there are still some errors/warnings.";
    }
    leaf compilation-result {
        type string;
        description
            "Result of the compilation explaining specifically what error or warnin
            g occurred.
            This is not existing if compilation status is PASSED.";
    }
    leaf reference {
        type string;
        description
            "A string that is used to specify a textual cross-reference to an exter
            nal document, either
            another module that defines related management information, or a docum
            ent that provides
            additional information relevant to this definition.";
    }
    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 string;
        default "1.0";
        description
            "The optional yang-version statement specifies which version of the
            YANG language was used in developing the module. The statement's
            argument is a string. If present, it MUST contain the value 1,
            which is the current YANG version and the default value.";
    }
    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 textual
    description of the organization(s) under whose auspices this module
    was developed.";
}
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 {
            value 0;
            description "If YANG file contains module.";
        }
        enum submodule {
            value 1;
            description "If YANG file contains sub-module.";
        }
    }
    description "Whether a file contains a YANG module or sub-module.";
}
leaf maturity-level {
    type enumeration {
        enum ratified {
            value 0;
            description
                "Maturity of a module that is fully approved (e.g., a standard).";
        }
        enum working-group {
            value 1;
            description
                "Maturity of a module that is actively being developed by a organiz
                ation towards ratification.";
```

```

    }
    enum individual {
        value 2;
        description
            "Maturity of a module that has been initially created, but has no o
fficial
            organization-level status.";
    }
}
description
    "The current maturity of the module with respect to the body that creat
ed it.
    This allows one to understand where the module is in its overall life
cycle.";
}
description
    "Grouping of YANG module metadata that extends the common list defined in
the YANG
    Module Library (RFC 7895).";
}
}
}
<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 hesitant 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 action.

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, <<http://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, <<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>>.

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-02 (work in progress), May 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://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.

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

Appendix B. Contributors

Contributors' Addresses

TBD

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
Intended status: Standards Track
Expires: October 5, 2018

J. Clarke
B. Claise
Cisco Systems, Inc.
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

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

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

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

This Internet-Draft will expire on October 5, 2018.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Status of Work and Open Issues	3
2. Learning from Experience	3
2.1. YANG Module Library	4
2.2. YANG Catalog Data Model	4
2.3. Module Sub-Tree	6
2.4. Compilation Information	8
2.5. Maturity Level	10
2.6. Generated From	11
2.7. Implementation	11
2.8. Vendor Sub-Tree	12
2.9. Regex Expression Differences	13
3. YANG Catalog Use Cases	14
3.1. YANG Search Metadata	14
3.2. Identify YANG Module Support in Devices	14
3.3. Identify The Backward Compatibility between YANG Module Revisions	14
4. YANG Catalog YANG module	17
5. Security Considerations	35
6. IANA Considerations	35
7. References	35
7.1. Normative References	35
7.2. Informative References	36
7.3. URIs	36
Appendix A. Acknowledgments	36
Appendix B. Changes From Previous Revisions	37
Authors' Addresses	38

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/contribute.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 subsequent 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 lifecycle 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 occurred.
    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 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 thus all vendor 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 manually 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 <edit-config>

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

    * 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/module

    * Use identities for protocols instead of an enumeration
    * 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 implementation 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
          le
          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 network
k device appliances,
                this will be the operating system version.  But for other YANG
G module
                implementation, this would be a version of appliance software
.  Ultimately,
                this should correspond to a version string that will be recognizable 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
network 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
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";
      }
    }
  }
}

```



```

        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 nodes.";
            }
            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 beyond the expires date, then expired
      should be true.";
  }
  description
    "Grouping of YANG module metadata that extends the common list defined in 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 metadata 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 manually 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 hesitant 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
Intended status: Experimental
Expires: October 1, 2017

A. Clemm
Huawei
E. Voit
J. Medved
Cisco Systems
March 30, 2017

Mounting YANG-Defined Information from Remote Datastores
draft-clemm-netmod-mount-06.txt

Abstract

This document introduces capabilities that allow YANG datastores to reference and incorporate information from remote datastores. This is accomplished by extending YANG with the ability to define mount points that reference data nodes in another YANG subtree, by subsequently allowing those data nodes to be accessed by client applications as if part of an alternative data hierarchy, and by providing the necessary means to manage and administer those mount points. Two flavors are defined: Alias-Mount allows to mount local subtrees, while Peer-Mount allows subtrees to reside on and be authoritatively owned by a remote server. YANG-Mount facilitates the development of applications that need to access data that transcends individual network devices while improving network-wide object consistency, or that require an aliasing capability to be able to create overlay structures for YANG data.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 1, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Overview	3
1.2. Examples	5
2. Definitions and Acronyms	7
3. Example scenarios	7
3.1. Network controller view	8
3.2. Consistent network configuration	10
4. Operating on mounted data	11
4.1. General principles	11
4.2. Data retrieval	12
4.3. Other operations	12
4.4. Other considerations	13
5. Data model structure	14
5.1. YANG mountpoint extensions	14
5.2. YANG structure diagrams	15
5.3. Mountpoint management	15
5.4. Caching	17
5.5. Other considerations	18
5.5.1. Authorization	18

5.5.2. Datastore qualification	18
5.5.3. Mount cascades	18
5.5.4. Implementation considerations	19
5.5.5. Modeling best practices	20
6. Datastore mountpoint YANG module	20
7. Security Considerations	28
8. Acknowledgements	28
9. Normative References	28
Appendix A. Example	30
Authors' Addresses	34

1. Introduction

1.1. Overview

This document introduces a new capability that allows YANG datastores [RFC7950] to incorporate and reference information from other YANG subtrees. The capability allows a client application to retrieve and have visibility of that YANG data as part of an alternative structure. This is provided by introducing a mountpoint concept. This concept allows to declare a YANG data node in a primary datastore to serve as a "mount point" under which a subtree with YANG data can be mounted. This way, data nodes from another subtree can be inserted into an alternative data hierarchy, arranged below local data nodes. To the user, this provides visibility to data from other subtrees, rendered in a way that makes it appear largely as if it were an integral part of the datastore. This enables users to retrieve local "native" as well as mounted data in integrated fashion, using e.g. Netconf [RFC6241] or Restconf [RFC8040] data retrieval primitives. The concept is reminiscent of concepts in a Network File System that allows to mount remote folders and make them appear as if they were contained in the local file system of the user's machine.

Two variants of YANG-Mount are introduced, which build on one another:

- o Alias-Mount allows mountpoints to reference a local YANG subtree residing on the same server. It provides effectively an aliasing capability, allowing for an alternative hierarchy and path for the same YANG data.
- o Peer-Mount allows mountpoints to reference a remote YANG subtree, residing on a different server. It can be thought of as an extension to Alias-Mount, in which a remote server can be specified. Peer-Mount allows a server to effectively provide a federated datastore, including YANG data from across the network.

In each case, mounted data is authoritatively owned by the server that it is a part of. Validation of integrity constraints apply to the authoritative copy; mounting merely provides a different view of the same data. It does not impose additional constraints on that same data; however, mounted data may be referred to from other data nodes. The mountpoint concept applies in principle to operations beyond data retrieval, i.e. to configuration, RPCs, and notifications. However, support for such operations involves additional considerations, for example if support for configuration transactions and locking (which might now apply across the network) were to be provided. While it is conceivable that additional capabilities for operations on mounted information are introduced at some point in time, their specification is beyond the scope of this specification.

YANG does provide means by which modules that have been separately defined can reference and augment one another. YANG also does provide means to specify data nodes that reference other data nodes. However, all the data is assumed to be instantiated as part of the same datastore, for example a datastore provided through a NETCONF server. Existing YANG mechanisms do not account for the possibility that some information that needs to be referred not only resides in a different subtree of the same datastore, or was defined in a separate module that is also instantiated in the same datastore, but that is genuinely part of a different datastore that is provided by a different server.

The ability to mount information from local and remote datastores is new and not covered by existing YANG mechanisms. Until now, management information provided in a datastore has been intrinsically tied to the same server and to a single data hierarchy. In contrast, the capability introduced in this specification allows the server to render alternative data hierarchies, and to represent information from remote systems as if it were its own and contained in its own local data hierarchy.

The capability of allowing the mounting of information from other subtrees is accomplished by a set of YANG extensions that allow to define such mount points. For this purpose, a new YANG module is introduced. The module defines the YANG extensions, as well as a data model that can be used to manage the mountpoints and mounting process itself. Only the mounting module and its server (i.e. the "receivers" or "consumers" of the mounted information) need to be aware of the concepts introduced here. Mounting is transparent to the "providers" of the mounted information and models that are being mounted; any data nodes or subtrees within any YANG model can be mounted.

Alias-Mount and Peer-Mount build on top of each other. It is possible for a server to support Alias-Mount but not Peer-Mount. In essence, Peer-Mount requires an additional parameter that is used to refer to the target system. This parameter does not need to be supported if only Alias-Mount is provided.

Finally, it should be mentioned that Alias-Mount and Peer-Mount are not to be confused with the ability to mount a schema, aka Schema Mount. A Schema Mount allows to instantiate an existing model definition underneath a mount point, not reference a set of YANG data that has already been instantiated somewhere else. In that sense, Schema-Mount resembles more a "grouping" concept that allows to reuse an existing definition in a new context, as opposed to referencing and incorporating existing instance information into a new context.

1.2. Examples

The ability to mount data from remote datastores is useful to address various problems that several categories of applications are faced with.

One category of applications that can leverage this capability are network controller applications that need to present a consolidated view of management information in datastores across a network. Controller applications are faced with the problem that in order to expose information, that information needs to be part of their own datastore. Today, this requires support of a corresponding YANG data module. In order to expose information that concerns other network elements, that information has to be replicated into the controller's own datastore in the form of data nodes that may mirror but are clearly distinct from corresponding data nodes in the network element's datastore. In addition, in many cases, a controller needs to impose its own hierarchy on the data that is different from the one that was defined as part of the original module. An example for this concerns interface data, both operational data (e.g. various types of interface statistics) and configuration data, such as defined in [RFC7223]. This data will be contained in a top-level container ("interfaces", in this particular case) in a network element datastore. The controller may need to provide its clients a view on interface data from multiple devices under its scope of control. One way of to do so would involve organizing the data in a list with separate list elements for each device. However, this in turn would require introduction of redundant YANG modules that effectively replicate the same interface data save for differences in hierarchy.

By directly mounting information from network element datastores, the controller does not need to replicate the same information from

multiple datastores, nor does it need to re-define any network element and system-level abstractions to be able to put them in the context of network abstractions. Instead, the subtree of the remote system is attached to the local mount point. Operations that need to access data below the mount point are in effect transparently redirected to remote system, which is the authoritative owner of the data. The mounting system does not even necessarily need to be aware of the specific data in the remote subtree. Optionally, caching strategies can be employed in which the mounting system prefetches data.

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters. When each network element maintains its own datastore with the same configurable settings, a single global change requires modifying the same information in many network elements across a network. In case of inconsistent configurations, network failures can result that are difficult to troubleshoot. In many cases, what is more desirable is the ability to configure such settings in a single place, then make them available to every network element. Today, this requires in general the introduction of specialized servers and configuration options outside the scope of NETCONF, such as RADIUS [RFC2866] or DHCP [RFC2131]. In order to address this within the scope of NETCONF and YANG, the same information would have to be redundantly modeled and maintained, representing operational data (mirroring some remote server) on some network elements and configuration data on a designated master. Either way, additional complexity ensues.

Instead of replicating the same global parameters across different datastores, the solution presented in this document allows a single copy to be maintained in a subtree of single datastore that is then mounted by every network element that requires awareness of these parameters. The global parameters can be hosted in a controller or a designated network element. This considerably simplifies the management of such parameters that need to be known across elements in a network and require global consistency.

It should be noted that for these and many other applications merely having a view of the remote information is sufficient. It allows to define consolidated views of information without the need for replicating data and models that have already been defined, to audit information, and to validate consistency of configurations across a network. Only retrieval operations are required; no operations that involve configuring remote data are involved.

2. Definitions and Acronyms

Data node: An instance of management information in a YANG datastore.

DHCP: Dynamic Host Configuration Protocol.

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

Datastore-push: A mechanism that allows a client to subscribe to updates from a datastore, which are then automatically pushed by the server to the client.

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

Mount client: The system at which the mount point resides, into which the remote subtree is mounted.

Mount point: A data node that receives the root node of the remote datastore being mounted.

Mount server: The server with which the mount client communicates and which provides the mount client with access to the mounted information. Can be used synonymously with mount target.

Mount target: A remote server whose datastore is being mounted.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

RADIUS: Remote Authentication Dial In User Service.

RPC: Remote Procedure Call

Remote datastore: A datastore residing at a remote node.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

3. Example scenarios

The following example scenarios outline some of the ways in which the ability to mount YANG datastores can be applied. Other mount topologies can be conceived in addition to the ones presented here.

3.1. Network controller view

Network controllers can use the mounting capability to present a consolidated view of management information across the network. This allows network controllers to expose network-wide abstractions, such as topologies or paths, multi-device abstractions, such as VRRP [RFC3768], and network-element specific abstractions, such as information about a network element's interfaces.

While an application on top of a controller could bypass the controller to access network elements directly for their element-specific abstractions, this would come at the expense of added inconvenience for the client application. In addition, it would compromise the ability to provide layered architectures in which access to the network by controller applications is truly channeled through the controller.

Without a mounting capability, a network controller would need to at least conceptually replicate data from network elements to provide such a view, incorporating network element information into its own controller model that is separate from the network element's, indicating that the information in the controller model is to be populated from network elements. This can introduce issues such as data inconsistency and staleness. Equally important, it would lead to the need to define redundant data models: one model that is implemented by the network element itself, and another model to be implemented by the network controller. This leads to poor maintainability, as analogous information has to be redundantly defined and implemented across different data models. In general, controllers cannot simply support the same modules as their network elements for the same information because that information needs to be put into a different context. This leads to "node"-information that needs to be instantiated and indexed differently, because there are multiple instances across different data stores.

For example, "system"-level information of a network element would most naturally be placed into a top-level container at that network element's datastore. At the same time, the same information in the context of the overall network, such as maintained by a controller, might better be provided in a list. For example, the controller might maintain a list with a list element for each network element, underneath which the network element's system-level information is contained. However, the containment structure of data nodes in a module, once defined, cannot be changed. This means that in the context of a network controller, a second module that repeats the same system-level information would need to be defined, implemented, and maintained. Any augmentations that add additional system-level information to the original module will likewise need to be

redundantly defined, once for the "system" module, a second time for the "controller" module.

By allowing a network controller to directly mount information from network element datastores, the controller does not need to replicate the same information from multiple datastores. Perhaps even more importantly, the need to re-define any network element and system-level abstractions just to be able to put them in the context of network abstractions is avoided. In this solution, a network controller's datastore mounts information from many network element datastores. For example, the network controller datastore (the "primary" datastore) could implement a list in which each list element contains a mountpoint. Each mountpoint mounts a subtree from a different network element's datastore. The data from the mounted subtrees is then accessible to clients of the primary datastore using the usual data retrieval operations.

This scenario is depicted in Figure 1. In the figure, M1 is the mountpoint for the datastore in Network Element 1 and M2 is the mountpoint for the datastore in Network Element 2. MDN1 is the mounted data node in Network Element 1, and MDN2 is the mounted data node in Network Element 2.

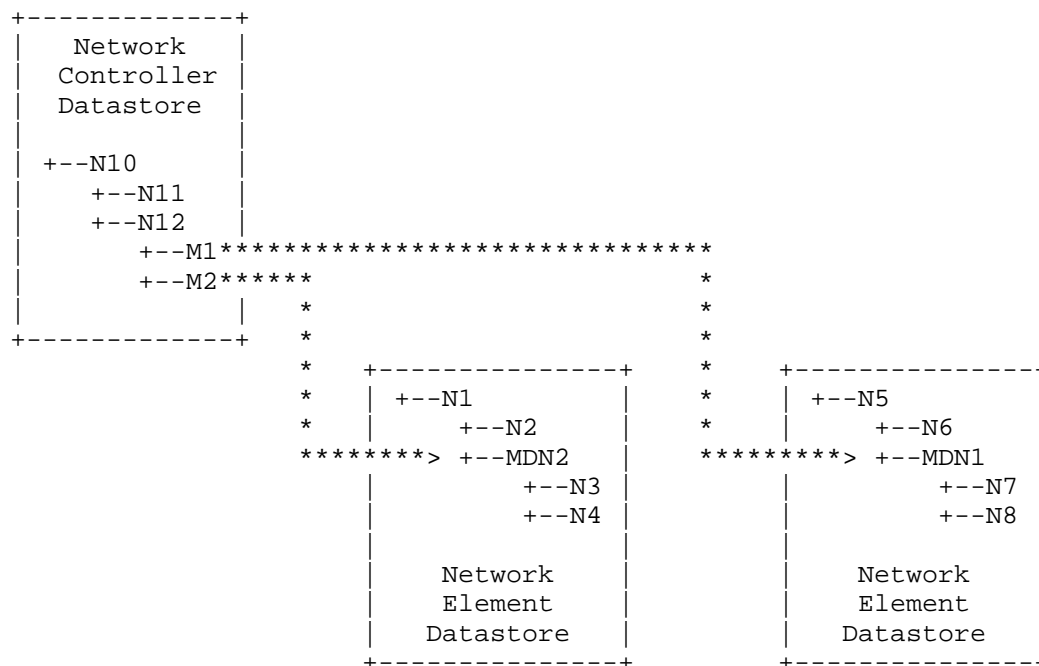


Figure 1: Network controller mount topology

3.2. Consistent network configuration

A second category of applications concerns decentralized networking applications that require globally consistent configuration of parameters that need to be known across elements in a network. Today, the configuration of such parameters is generally performed on a per network element basis, which is not only redundant but, more importantly, error-prone. Inconsistent configurations lead to erroneous network behavior that can be challenging to troubleshoot.

Using the ability to mount information from remote datastores opens up a new possibility for managing such settings. Instead of replicating the same global parameters across different datastores, a single copy is maintained in a subtree of single datastore. This datastore can be hosted in a controller or a designated network element. The subtree is subsequently mounted by every network element that requires access to these parameters.

In many ways, this category of applications is an inverse of the previous category: Whereas in the network controller case data from many different datastores would be mounted into the same datastore with multiple mountpoints, in this case many elements, each with their own datastore, mount the same remote datastore, which is then mounted by many different systems.

The scenario is depicted in Figure 2. In the figure, M1 is the mountpoint for the Network Controller datastore in Network Element 1 and M2 is the mountpoint for the Network Controller datastore in Network Element 2. MDN is the mounted data node in the Network Controller datastore that contains the data nodes that represent the shared configuration settings. (Note that there is no reason why the Network Controller Datastore in this figure could not simply reside on a network element itself; the division of responsibilities is a logical one.

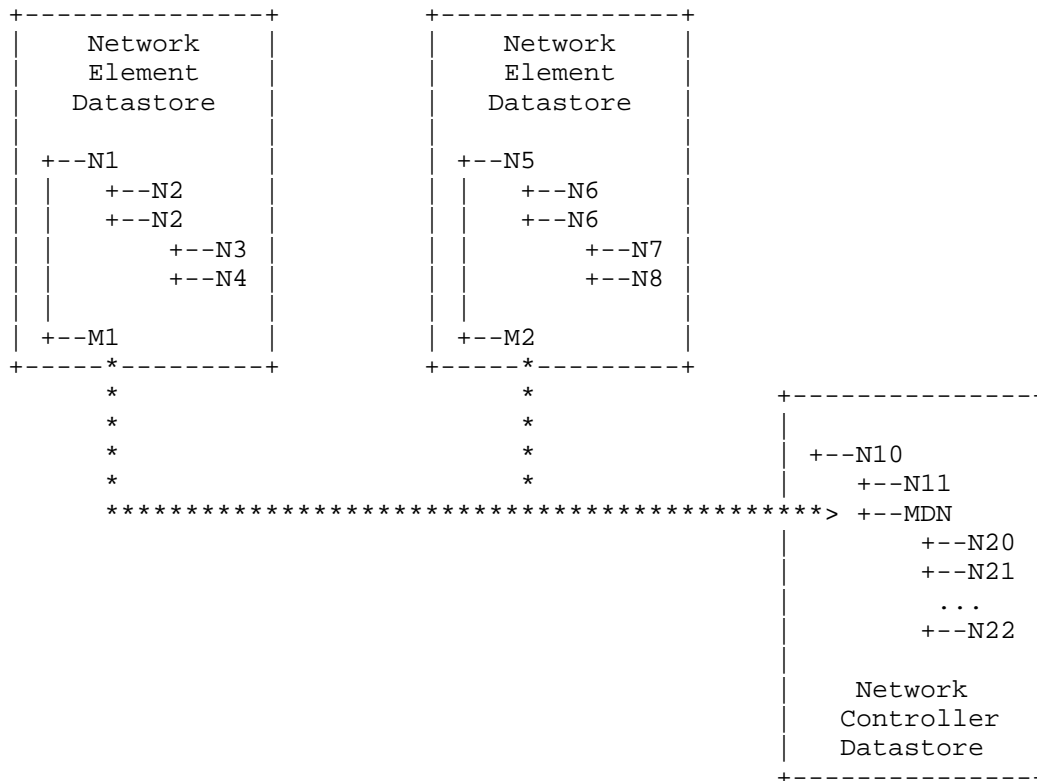


Figure 2: Distributed config settings topology

4. Operating on mounted data

This section provides a rough illustration of the operations flow involving mounted datastores.

4.1. General principles

The first thing that should be noted about these operations flows concerns the fact that a mount client essentially constitutes a special management application that interacts with a subtree to render the data of that subtree as an alternative tree hierarchy. In the case of Alias-Mount, both original and alternative tree are maintained by the same server, which in effect provides alternative paths to the same data. In the case of Peer-Mount, the mount client constitutes in effect another application, with the remote system remaining the authoritative owner of the data. While it is conceivable that the remote system (or an application that proxies for the remote system) provides certain functionality to facilitate

the specific needs of the mount client to make it more efficient, the fact that another system decides to expose a certain "view" of that data is fundamentally not the remote system's concern.

When a client application makes a request to a server that involves data that is mounted from a remote system, the server will effectively act as a proxy to the remote system on the client application's behalf. It will extract from the client application request the portion that involves the mounted subtree from the remote system. It will strip that portion of the local context, i.e. remove any local data paths and insert the data path of the mounted remote subtree, as appropriate. The server will then forward the transposed request to the remote system that is the authoritative owner of the mounted data, acting itself as a client to the remote server. Upon receiving the reply, the server will transpose the results into the local context as needed, for example map the data paths into the local data tree structure, and combine those results with the results of the remainder portion of the original request.

4.2. Data retrieval

Data retrieval operations are the only category of operations that is supported for peer-mounted information. In that case, a Netconf "get" or "get-configuration" operation might be applied on a subtree whose scope includes a mount point. When resolving the mount point, the server issues its own "get" or "get-configuration" request against the remote system's subtree that is attached to the mount point. The returned information is then inserted into the data structure that is in turn returned to the client that originally invoked the request.

4.3. Other operations

The fact that only data retrieval operations are the only category of operations that are supported for peer-mounted information does not preclude other operations to be applied to datastore subtrees that contain mountpoints and peer-mounted information. Peer-mounted information is simply transparent to those operations. When an operation is applied to a subtree which includes mountpoints, mounted information is ignored for purposes of the operation. For example, for a Netconf "edit-config" operation that includes a subtree with a mountpoint, a server will ignore the data under the mountpoint and apply the operation only to the local configuration. Mounted data is "read-only" data. The server does not even need to return an error message that the operation could not be applied to mounted data; the mountpoint is simply ignored.

In principle, it is conceivable that operations other than data-retrieval are applied to mounted data as well. For example, an operation to edit configuration information might expect edits to be applied to remote systems as part of the operation, where the edited subtree involves mounted information. However, editing of information and "writing through" to remote systems potentially involves significant complexity, particularly if transactions and locking across multiple configuration items are involved. Support for such operations will require additional capabilities, specification of which is beyond the scope of this specification.

Likewise, YANG-Mount does not extend towards RPCs that are defined as part of YANG modules whose contents is being mounted. Support for RPCs that involve mounted portions of the datastore, while conceivable, would require introduction of an additional capability, whose definition is outside the scope of this specification.

By the same token, YANG-Mount does not extend towards notifications. It is conceivable to offer such support in the future using a separate capability, definition of which is once again outside the scope of this specification.

4.4. Other considerations

Since mounting of information typically involves communication with a remote system, there is a possibility that the remote system will not respond within a certain amount of time, that connectivity is lost, or that other errors occur. Accordingly, the ability to mount datastores also involves mountpoint management, which includes the ability to configure timeouts, retries, and management of mountpoint state (including dynamic addition removal of mountpoints). Mountpoint management will be discussed in section Section 5.3.

It is expected that some implementations will introduce caching schemes. Caching can increase performance and efficiency in certain scenarios (for example, in the case of data that is frequently read but that rarely changes), but increases implementation complexity. Caching is not required for YANG-mount to work - in which case access to mounted information is "on-demand", in which the authoritative data node always gets accessed. Whether to perform caching is a local implementation decision.

When caching is introduced, it can benefit from the ability to subscribe to updates on remote data by remote servers. Some optimizations to facilitate caching support will be discussed in section Section 5.4.

5. Data model structure

5.1. YANG mountpoint extensions

At the center of the module is a set of YANG extensions that allow to define a mountpoint.

- o The first extension, "mountpoint", is used to declare a mountpoint. The extension takes the name of the mountpoint as an argument.
- o The second extension, "subtree", serves as substatement underneath a mountpoint statement. It takes an argument that defines the root node of the datastore subtree that is to be mounted, specified as string that contains a path expression. This extension is used to define mountpoints for Alias-Mount, as well as Peer-Mount.
- o The third extension, "target", also serves as a substatement underneath a mountpoint statement. It is used for Peer-Mount and takes an argument that identifies the target system. The argument is a reference to a data node that contains the information that is needed to identify and address a remote server, such as an IP address, a host name, or a URI [RFC3986].

A mountpoint **MUST** be contained underneath a container. Future revisions might allow for mountpoints to be contained underneath other data nodes, such as lists, leaf-lists, and cases. However, to keep things simple, at this point mounting is only allowed directly underneath a container.

Only a single data node can be mounted at one time. While the mount target could refer to any data node, it is recommended that as a best practice, the mount target **SHOULD** refer to a container. It is possible to maintain e.g. a list of mount points, with each mount point each of which has a mount target an element of a remote list. However, to avoid unnecessary proliferation of the number of mount points and associated management overhead, when data from lists or leaf-lists is to be mounted, a container containing the list respectively leaf-list **SHOULD** be mounted instead of individual list elements.

It is possible for a mounted datastore to contain another mountpoint, thus leading to several levels of mount indirections. However, mountpoints **MUST NOT** introduce circular dependencies. In particular, a mounted datastore **MUST NOT** contain a mountpoint which specifies the mounting datastore as a target and a subtree which contains as root node a data node that in turn contains the original mountpoint.

Whenever a mount operation is performed, this condition mountpoint.
Whenever a mount operation is performed, this condition MUST be
validated by the mount client.

5.2. YANG structure diagrams

YANG data model structure overviews have proven very useful to convey the "Big Picture". It would be useful to indicate in YANG data model structure overviews the fact that a given data node serves as a mountpoint. We propose for this purpose also a corresponding extension to the structure representation convention. Specifically, we propose to prefix the name of the mounting data node with upper-case 'M'.

```
rw network
+-- rw nodes
  +-- rw node [node-ID]
    +-- rw node-ID
    +-- M node-system-info
```

5.3. Mountpoint management

The YANG module contains facilities to manage the mountpoints themselves.

For this purpose, a list of the mountpoints is introduced. Each list element represents a single mountpoint. It includes an identification of the mount target, i.e. the remote system hosting the remote datastore and a definition of the subtree of the remote data node being mounted. It also includes monitoring information about current status (indicating whether the mount has been successful and is operational, or whether an error condition applies such as the target being unreachable or referring to an invalid subtree).

In addition to the list of mountpoints, a set of global mount policy settings allows to set parameters such as mount retries and timeouts.

Each mountpoint list element also contains a set of the same configuration knobs, allowing administrators to override global mount policies and configure mount policies on a per-mountpoint basis if needed.

There are two ways how mounting occurs: automatic (dynamically performed as part of system operation) or manually (administered by a user or client application). A separate mountpoint-origin object is used to distinguish between manually configured and automatically populated mountpoints.

Whether mounting occurs automatically or needs to be manually configured by a user or an application can depend on the mountpoint being defined, i.e. the semantics of the model.

When configured automatically, mountpoint information is automatically populated by the datastore that implements the mountpoint. The precise mechanisms for discovering mount targets and bootstrapping mount points are provided by the mount client infrastructure and outside the scope of this specification. Likewise, when a mountpoint should be deleted and when it should merely have its mount-status indicate that the target is unreachable is a system-specific implementation decision.

Manual mounting consists of two steps. In a first step, a mountpoint is manually configured by a user or client application through administrative action. Once a mountpoint has been configured, actual mounting occurs through an RPCs that is defined specifically for that purpose. To unmount, a separate RPC is invoked; mountpoint configuration information needs to be explicitly deleted. Manual mounting can also be used to override automatic mounting, for example to allow an administrator to set up or remove a mountpoint.

It should be noted that mountpoint management does not allow users to manually "extend" the model, i.e. simply add a subtree underneath some arbitrary data node into a datastore, without a supporting mountpoint defined in the model to support it. A mountpoint definition is a formal part of the model with well-defined semantics. Accordingly, mountpoint management does not allow users to dynamically "extend" the data model itself. It allows users to populate the datastore and mount structure within the confines of a model that has been defined prior.

The structure of the mountpoint management data model is depicted in the following figure, where brackets enclose list keys, "rw" means configuration, "ro" operational state data, and "?" designates optional nodes. Parentheses enclose choice and case nodes. The figure does not depict all definitions; it is intended to illustrate the overall structure.

```

module: ietf-mount
  +--rw mount-server-mgmt {mount-server-mgmt}?
    +--rw mountpoints
      +--rw mountpoint* [mountpoint-id]
        +--rw mountpoint-id      string
        +--ro mountpoint-origin? enumeration
        +--rw subtree-ref        subtree-ref
        +--rw mount-target
          +--rw (target-address-type)
            +--:(IP)
              | +--rw target-ip?      inet:ip-address
            +--:(URI)
              | +--rw uri?            inet:uri
            +--:(host-name)
              | +--rw hostname?       inet:host
            +--:(node-ID)
              | +--rw node-info-ref?  subtree-ref
            +--:(other)
              +--rw opaque-target-ID? string
          +--ro mount-status?        mount-status
        +--rw manual-mount?         empty
        +--rw retry-timer?          uint16
        +--rw number-of-retries?    uint8
      +--rw global-mount-policies
        +--rw manual-mount?         empty
        +--rw retry-timer?          uint16
        +--rw number-of-retries?    uint8

```

5.4. Caching

Under certain circumstances, it can be useful to maintain a cache of remote information. Instead of accessing the remote system, requests are served from a copy that is locally maintained. This is particularly advantageous in cases where data is slow changing, i.e. when there are many more "read" operations than changes to the underlying data node, and in cases when a significant delay were incurred when accessing the remote system, which might be prohibitive for certain applications. Examples of such applications are applications that involve real-time control loops requiring response times that are measured in milliseconds. However, as data nodes that are mounted from an authoritative datastore represent the "golden copy", it is important that any modifications are reflected as soon as they are made.

It is a local implementation decision of mount clients whether to cache information once it has been fetched. However, in order to support more powerful caching schemes, it becomes necessary for the mount server to "push" information proactively. For this purpose, it

is useful for the mount client to subscribe for updates to the mounted information at the mount server. A corresponding mechanism that can be leveraged for this purpose is specified in draft-ietf-netconf-yang-push-05.

Note that caching large mountpoints can be expensive. Therefore limiting the amount of data unnecessarily passed when mounting near the top of a YANG subtree is important. For these reasons, an ability to specify a particular caching strategy in conjunction with mountpoints can be desirable, including the ability to exclude certain nodes and subtrees from caching. According capabilities may be introduced in a future version of this draft.

5.5. Other considerations

5.5.1. Authorization

Access to mounted information is subject to authorization rules. To the mounted system, a mounting client will in general appear like any other client. Authorization privileges for remote mounting clients need to be specified through NACM (NETCONF Access Control Model) [RFC6536].

5.5.2. Datastore qualification

It is conceivable to differentiate between different datastores on the remote server, that is, to designate the name of the actual datastore to mount, e.g. "running" or "startup". However, for the purposes of this spec, we assume that the datastore to be mounted is generally implied. Mounted information is treated as analogous to operational data; in general, this means the running or "effective" datastore is the target. That said, the information which targets to mount does constitute configuration and can hence be part of a startup or candidate datastore.

5.5.3. Mount cascades

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount target that directly or indirectly contains the originating mountpoint. As part of a mount operation, the mount points of the mounted system need to be checked accordingly.

5.5.4. Implementation considerations

Implementation specifics are outside the scope of this specification. That said, the following considerations apply:

Systems that wish to mount information from remote datastores need to implement a mount client. The mount client communicates with a remote system to access the remote datastore. To do so, there are several options:

- o The mount client acts as a NETCONF client to a remote system. Alternatively, another interface to the remote system can be used, such as a REST API using JSON encodings, as specified in [RFC7951]. --> Either way, to the remote system, the mount client constitutes essentially a client application like any other. The mount client in effect IS a special kind of client application.
- o The mount client communicates with a remote mount server through a separate protocol. The mount server is deployed on the same system as the remote NETCONF datastore and interacts with it through a set of local APIs.
- o The mount client communicates with a remote mount server that acts as a NETCONF client proxy to a remote system, on the client's behalf. The communication between mount client and remote mount server might involve a separate protocol, which is translated into NETCONF operations by the remote mount server.

It is the responsibility of the mount client to manage the association with the target system, e.g. validate it is still reachable by maintaining a permanent association, perform reachability checks in case of a connectionless transport, etc.

It is the responsibility of the mount client to manage the mountpoints. This means that the mount client needs to populate the mountpoint monitoring information (e.g. keep mount-status up to data and determine in the case of automatic mounting when to add and remove mountpoint configuration). In the case of automatic mounting, the mount client also interacts with the mountpoint discovery and bootstrap process.

The mount client needs to also participate in servicing datastore operations involving mounted information. An operation requested involving a mountpoint is relayed by the mounting system's infrastructure to the mount client. For example, a request to retrieve information from a datastore leads to an invocation of an internal mount client API when a mount point is reached. The mount client then relays a corresponding operation to the remote datastore.

It subsequently relays the result along with any responses back to the invoking infrastructure, which then merges the result (e.g. a retrieved subtree with the rest of the information that was retrieved) as needed. Relaying the result may involve the need to transpose error response codes in certain corner cases, e.g. when mounted information could not be reached due to loss of connectivity with the remote server, or when a configuration request failed due to validation error.

5.5.5. Modeling best practices

There is a certain amount of overhead associated with each mount point. The mount point needs to be managed and state maintained. Data subscriptions need to be maintained. Requests including mounted subtrees need to be decomposed and responses from multiple systems combined.

For those reasons, as a general best practice, models that make use of mount points SHOULD be defined in a way that minimizes the number of mountpoints required. Finely granular mounts, in which multiple mountpoints are maintained with the same remote system, each containing only very small data subtrees, SHOULD be avoided. For example, lists SHOULD only contain mountpoints when individual list elements are associated with different remote systems. To mount data from lists in remote datastores, a container node that contains all list elements SHOULD be mounted instead of mounting each list element individually. Likewise, instead of having mount points refer to nodes contained underneath choices, a mountpoint should refer to a container of the choice.

6. Datastore mountpoint YANG module

```
<CODE BEGINS>
file "ietf-mount@2017-03-30.yang"
module ietf-mount {
  namespace "urn:ietf:params:xml:ns:yang:ietf-mount";
  prefix mnt;

  import ietf-inet-types {
    prefix inet;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
  contact
    "WG Web:  <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>
```


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

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

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

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

Editor: Eric Voit
<mailto:evoit@cisco.com>";

description

"This module provides a set of YANG extensions and definitions that can be used to mount information from remote datastores.";

```
revision 2017-03-30 {  
  description  
    "Initial revision.";  
  reference  
    "draft-clemm-netmod-mount-06.txt";  
}
```

```
extension mountpoint {  
  argument name;  
  description  
    "This YANG extension is used to mount data from another  
    subtree in place of the node under which this YANG extension  
    statement is used."
```

This extension takes one argument which specifies the name of the mountpoint.

This extension can occur as a substatement underneath a container statement, a list statement, or a case statement. As a best practice, it SHOULD occur as statement only underneath a container statement, but it MAY also occur underneath a list or a case statement.

The extension can take two parameters, target and subtree, each defined as their own YANG extensions.

For Alias-Mount, a mountpoint statement MUST contain a subtree statement for the mountpoint definition to be valid. For Peer-Mount, a mountpoint statement MUST contain both a target and a subtree substatement for the mountpoint

definition to be valid.

The subtree SHOULD be specified in terms of a data node of type 'mnt:subtree-ref'. The targeted data node MUST represent a container.

The target system MAY be specified in terms of a data node that uses the grouping 'mnt:mount-target'. However, it can be specified also in terms of any other data node that contains sufficient information to address the mount target, such as an IP address, a host name, or a URI.

It is possible for the mounted subtree to in turn contain a mountpoint. However, circular mount relationships MUST NOT be introduced. For this reason, a mounted subtree MUST NOT contain a mountpoint that refers back to the mounting system with a mount target that directly or indirectly contains the originating mountpoint.";

}

```
extension target {
  argument target-name;
  description
    "This YANG extension is used to perform a Peer-Mount.
    It is used to specify a remote target system from which to
    mount a datastore subtree. This YANG
    extension takes one argument which specifies the remote
    system. In general, this argument will contain the name of
    a data node that contains the remote system information. It
    is recommended that the reference data node uses the
    mount-target grouping that is defined further below in this
    module.
```

This YANG extension can occur only as a substatement below a mountpoint statement. It MUST NOT occur as a substatement below any other YANG statement.";

}

```
extension subtree {
  argument subtree-path;
  description
    "This YANG extension is used to specify a subtree in a
    datastore that is to be mounted. This YANG extension takes
    one argument which specifies the path to the root of the
    subtree. The root of the subtree SHOULD represent an
    instance of a YANG container. However, it MAY represent
    also another data node.
```

```
    This YANG extension can occur only as a substatement below
    a mountpoint statement. It MUST NOT occur as a substatement
    below any other YANG statement.";
}

feature mount-server-mgmt {
  description
    "Provide additional capabilities to manage remote mount
    points";
}

typedef mount-status {
  type enumeration {
    enum "ok" {
      description
        "Mounted";
    }
    enum "no-target" {
      description
        "The argument of the mountpoint does not define a
        target system";
    }
    enum "no-subtree" {
      description
        "The argument of the mountpoint does not define a
        root of a subtree";
    }
    enum "target-unreachable" {
      description
        "The specified target system is currently
        unreachable";
    }
    enum "mount-failure" {
      description
        "Any other mount failure";
    }
    enum "unmounted" {
      description
        "The specified mountpoint has been unmounted as the
        result of a management operation";
    }
  }
  description
    "This type is used to represent the status of a
    mountpoint.";
}

typedef subtree-ref {
```

```
type string;
description
  "This string specifies a path to a datanode. It corresponds
  to the path substatement of a leafref type statement. Its
  syntax needs to conform to the corresponding subset of the
  XPath abbreviated syntax. Contrary to a leafref type,
  subtree-ref allows to refer to a node in a remote datastore.
  Also, a subtree-ref refers only to a single node, not a list
  of nodes."
}

grouping mount-monitor {
  description
    "This grouping contains data nodes that indicate the
    current status of a mount point."
  leaf mount-status {
    type mount-status;
    config false;
    description
      "Indicates whether a mountpoint has been successfully
      mounted or whether some kind of fault condition is
      present."
  }
}

grouping mount-target {
  description
    "This grouping contains data nodes that can be used to
    identify a remote system from which to mount a datastore
    subtree."
  container mount-target {
    description
      "A container is used to keep mount target information
      together."
    choice target-address-type {
      mandatory true;
      description
        "Allows to identify mount target in different ways,
        i.e. using different types of addresses."
      case IP {
        leaf target-ip {
          type inet:ip-address;
          description
            "IP address identifying the mount target."
        }
      }
      case URI {
        leaf uri {
```

```

        type inet:uri;
        description
            "URI identifying the mount target";
    }
}
case host-name {
    leaf hostname {
        type inet:host;
        description
            "Host name of mount target.";
    }
}
case node-ID {
    leaf node-info-ref {
        type subtree-ref;
        description
            "Node identified by named subtree.";
    }
}
case other {
    leaf opaque-target-ID {
        type string;
        description
            "Catch-all; could be used also for mounting
            of data nodes that are local.";
    }
}
}
}
}

grouping mount-policies {
    description
        "This grouping contains data nodes that allow to configure
        policies associated with mountpoints.";
    leaf manual-mount {
        type empty;
        description
            "When present, a specified mountpoint is not
            automatically mounted when the mount data node is
            created, but needs to be mounted via specific RPC
            invocation.";
    }
    leaf retry-timer {
        type uint16;
        units "seconds";
        description
            "When specified, provides the period after which

```

```
        mounting will be automatically reattempted in case of a
        mount status of an unreachable target";
    }
    leaf number-of-retries {
        type uint8;
        description
            "When specified, provides a limit for the number of
            times for which retries will be automatically
            attempted";
    }
}

rpc mount {
    description
        "This RPC allows an application or administrative user to
        perform a mount operation.  If successful, it will result in
        the creation of a new mountpoint.";
    input {
        leaf mountpoint-id {
            type string {
                length "1..32";
            }
            description
                "Identifier for the mountpoint to be created.
                The mountpoint-id needs to be unique;
                if the mountpoint-id of an existing mountpoint is
                chosen, an error is returned.";
        }
    }
    output {
        leaf mount-status {
            type mount-status;
            description
                "Indicates if the mount operation was successful.";
        }
    }
}

rpc unmount {
    description
        "This RPC allows an application or administrative user to
        unmount information from a remote datastore.  If successful,
        the corresponding mountpoint will be removed from the
        datastore.";
    input {
        leaf mountpoint-id {
            type string {
                length "1..32";
            }
        }
    }
}
```

```
        description
            "Identifies the mountpoint to be unmounted.";
    }
}
output {
    leaf mount-status {
        type mount-status;
        description
            "Indicates if the unmount operation was successful.";
    }
}
}
container mount-server-mgmt {
    if-feature mount-server-mgmt;
    description
        "Contains information associated with managing the
        mountpoints of a datastore.";
    container mountpoints {
        description
            "Keep the mountpoint information consolidated
            in one place.";
        list mountpoint {
            key "mountpoint-id";
            description
                "There can be multiple mountpoints.
                Each mountpoint is represented by its own
                list element.";
            leaf mountpoint-id {
                type string {
                    length "1..32";
                }
                description
                    "An identifier of the mountpoint.
                    RPC operations refer to the mountpoint
                    using this identifier.";
            }
            leaf mountpoint-origin {
                type enumeration {
                    enum "client" {
                        description
                            "Mountpoint has been supplied and is
                            manually administered by a client";
                    }
                    enum "auto" {
                        description
                            "Mountpoint is automatically
                            administered by the server";
                    }
                }
            }
        }
    }
}
```

```
    }
    config false;
    description
      "This describes how the mountpoint came
       into being.";
  }
  leaf subtree-ref {
    type subtree-ref;
    mandatory true;
    description
      "Identifies the root of the subtree in the
       target system that is to be mounted.";
  }
  uses mount-target;
  uses mount-monitor;
  uses mount-policies;
}
}
container global-mount-policies {
  description
    "Provides mount policies applicable for all mountpoints,
     unless overridden for a specific mountpoint.";
  uses mount-policies;
}
}
```

<CODE ENDS>

7. Security Considerations

TBD

8. Acknowledgements

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Tony Tkacik, Ambika Tripathy, Robert Varga, Prabhakara Yellai, Shashi Kumar Bansal, Lukas Sedlak, and Benoit Claise.

9. Normative References

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <<http://www.rfc-editor.org/info/rfc2131>>.

- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, DOI 10.17487/RFC2866, June 2000, <<http://www.rfc-editor.org/info/rfc2866>>.
- [RFC3768] Hinden, R., Ed., "Virtual Router Redundancy Protocol (VRRP)", RFC 3768, DOI 10.17487/RFC3768, April 2004, <<http://www.rfc-editor.org/info/rfc3768>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [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>>.
- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <<http://www.rfc-editor.org/info/rfc7923>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.
- [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

In the following example, we are assuming the use case of a network controller that wants to provide a controller network view to its client applications. This view needs to include network abstractions that are maintained by the controller itself, as well as certain information about network devices where the network abstractions tie in with element-specific information. For this purpose, the network controller leverages the mount capability specified in this document and presents a fictitious Controller Network YANG Module that is depicted in the outlined structure below. The example illustrates how mounted information is leveraged by the mounting datastore to provide an additional level of information that ties together network and device abstractions, which could not be provided otherwise without introducing a (redundant) model to replicate those device abstractions

```
rw controller-network
+-- rw topologies
|   +-- rw topology [topo-id]
|       +-- rw topo-id                node-id
|       +-- rw nodes
|           +-- rw node [node-id]
|               +-- rw node-id                node-id
|               +-- rw supporting-ne          network-element-ref
|               +-- rw termination-points
|                   +-- rw term-point [tp-id]
|                       +-- tp-id                tp-id
|                       +-- ifref                mountedIfRef
|       +-- rw links
|           +-- rw link [link-id]
|               +-- rw link-id                link-id
|               +-- rw source                  tp-ref
|               +-- rw dest                    tp-ref
+-- rw network-elements
    +-- rw network-element [element-id]
        +-- rw element-id                element-id
        +-- rw element-address
        |   +-- ...
        +-- M interfaces
```

The controller network model consists of the following key components:

- o A container with a list of topologies. A topology is a graph representation of a network at a particular layer, for example, an IS-IS topology, an overlay topology, or an Openflow topology. Specific topology types can be defined in their own separate YANG

modules that augment the controller network model. Those augmentations are outside the scope of this example

- o An inventory of network elements, along with certain information that is mounted from each element. The information that is mounted in this case concerns interface configuration information. For this purpose, each list element that represents a network element contains a corresponding mountpoint. The mountpoint uses as its target the network element address information provided in the same list element
- o Each topology in turn contains a container with a list of nodes. A node is a network abstraction of a network device in the topology. A node is hosted on a network element, as indicated by a network-element leafref. This way, the "logical" and "physical" aspects of a node in the network are cleanly separated.
- o A node also contains a list of termination points that terminate links. A termination point is implemented on an interface. Therefore, it contains a leafref that references the corresponding interface configuration which is part of the mounted information of a network element. Again, the distinction between termination points and interfaces provides a clean separation between logical concepts at the network topology level and device-specific concepts that are instantiated at the level of a network element. Because the interface information is mounted from a different datastore and therefore occurs at a different level of the containment hierarchy than it would if it were not mounted, it is not possible to use the interface-ref type that is defined in YANG data model for interface management [] to allow the termination point refer to its supporting interface. For this reason, a new type definition "mountedIfRef" is introduced that allows to refer to interface information that is mounted and hence has a different path.
- o Finally, a topology also contains a container with a list of links. A link is a network abstraction that connects nodes via node termination points. In the example, directional point-to-point links are depicted in which one node termination point serves as source, another as destination.

The following is a YANG snippet of the module definition which makes use of the mountpoint definition.

```

<CODE BEGINS>
module controller-network {
    namespace "urn:cisco:params:xml:ns:yang:controller-network";
    // example only, replace with IANA namespace when assigned
    prefix cn;
    import mount {
        prefix mnt;
    }
    import interfaces {
        prefix if;
    }
    ...
    typedef mountedIfRef {
        type leafref {
            path "/cn:controller-network/cn:network-elements/"
              + "cn:network-element/cn:interfaces/if:interface/if:name";
            // cn:interfaces corresponds to the mountpoint
        }
    }
    ...
    list termination-point {
        key "tp-id";
        ...
        leaf ifref {
            type mountedIfRef;
        }
        ...
        list network-element {
            key "element-id";
            leaf element-id {
                type element-ID;
            }
            container element-address {
                ... // choice definition that allows to specify
                // host name,
                // IP addresses, URIs, etc
            }
            mnt:mountpoint "interfaces" {
                mnt:target "./element-address";
                mnt:subtree "/if:interfaces";
            }
            ...
        }
    }
    ...
<CODE ENDS>

```

Finally, the following contains an XML snippet of instantiated YANG information. We assume three datastores: NE1 and NE2 each have a

datastore (the mount targets) that contains interface configuration data, which is mounted into NC's datastore (the mount client).

Interface information from NE1 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/1</name>
    <name>ethernetCsmacd</type>
    <location>1/1</location>
  </interface>
</interfaces>
```

Interface information from NE2 datastore:

```
<interfaces>
  <interface>
    <name>fastethernet-1/0</name>
    <name>ethernetCsmacd</type>
    <location>1/0</location>
  </interface>
  <interface>
    <name>fastethernet-1/2</name>
    <name>ethernetCsmacd</type>
    <location>1/2</location>
  </interface>
</interfaces>
```

NC datastore with mounted interface information from NE1 and NE2:

```
<controller-network>
...
<network-elements>
  <network-element>
    <element-id>NE1</element-id>
    <element-address> .... </element-address>
    <interfaces>
      <if:interface>
        <if:name>fastethernet-1/0</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/0</if:location>
      </if:interface>
      <if:interface>
        <if:name>fastethernet-1/1</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/1</if:location>
      </if:interface>
    </interfaces>
  </network-element>
  <network-element>
    <element-id>NE2</element-id>
    <element-address> .... </element-address>
    <interfaces>
      <if:interface>
        <if:name>fastethernet-1/0</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/0</if:location>
      </if:interface>
      <if:interface>
        <if:name>fastethernet-1/2</if:name>
        <if:type>ethernetCsmacd</if:type>
        <if:location>1/2</if:location>
      </if:interface>
    </interfaces>
  </network-element>
</network-elements>
...
</controller-network>
```

Authors' Addresses

Alexander Clemm
Huawei

E-Mail: ludwig@clemm.org

Eric Voit
Cisco Systems

EMail: evoit@cisco.com

Jan Medved
Cisco Systems

EMail: jmedved@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 10, 2017

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
May 9, 2017

Guidelines for YANG Module Authors (NMDA)
draft-dsdt-nmda-guidelines-01

Abstract

The "Network Management Datastore Architecture" (NMDA) adds the ability to inspect the current operational values for configuration, allowing clients to use identical paths for retrieving the configured values and the operational values. This change will simplify models and help modelers, but will create a period of transition as NMDA becomes a standard and is widely implemented. During that interim, the guidelines given in this document should help modelers find an optimal path forward.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 10, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Keywords	2
1.2. Terminology	2
1.3. Executive Summary	3
1.4. Background	3
1.5. Network Management Datastores Architecture	5
2. Guidelines for YANG Modelers	5
3. IANA Considerations	8
4. Security Considerations	8
5. Informative References	8
Authors' Addresses	9

1. Introduction

This document provides advice and guidelines to help modelers plan for the emerging "Network Management Datastore Architecture" (NMDA) [I-D.ietf-netmod-revised-datastores]. This architecture 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, enabling data models to be written in a network management protocol agnostic way.

1.1. Keywords

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].

1.2. Terminology

This document uses the terminology defined by the NMDA [I-D.ietf-netmod-revised-datastores].

1.3. Executive Summary

The Network Management Datastore Architecture (NMDA) addresses the so called "OpState problem" that has been the subject of much discussion in the IETF. NMDA is still in development and there will be a transition period before NMDA solutions are universally available.

These guidelines are aimed to enable the creation of models that can take advantage of the NMDA, while pragmatically allowing those models to be used with the existing network configuration protocol implementations.

It is the strong recommendation that models SHOULD move as quickly as possible to the NMDA. The specific approach to be taken for models being developed now and during the NMDA transition period should be based on both the expected usage and the maturity of the data model.

1. New models and models that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA-compatible.
2. Models that require immediate support for "in use" and "system created" information SHOULD be structured for NMDA. A non-NMDA version of these models SHOULD also be published, using either an existing model or a model created either by hand or with suitable tools that support current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative appendix. The use of the non-NMDA model will allow temporary bridging of the time period until NMDA implementations are available.

Additional details on these guidelines can be found below, notably in Section 2.

1.4. Background

NETCONF ([RFC6241]) was developed with a focus on configuration data, and has unfortunate gaps in its treatment of operational data. The <get-config> operation can return configuration data (defined as nodes with "config true") stored in <running>. This data is typically the only data created by CLI users and NETCONF clients. The <get> operation is defined as returning all the data on the device, including the contents of <running>, as well as any operational state data. While the NETCONF design envisioned models merging "config false" nodes with the contents of running, there are two issues involved.

First, the desire of clients to see the true operational ("in use") value of configuration data resulted in the need for data models to have two distinct leafs, one to show the configured value and the other to show the operational value. An example would be the speed of an interface, where the configured value may not be the value that is currently used.

Second, devices often have "system created" resources that exist as operational data even when there is no corresponding configuration data. An example would be built-in networking interfaces that always appear in operational data.

A similar situation to the second issue discussed above exists while the device is processing configuration data changes. When configuration data is deleted, the operational data will continue to exist during the time period in which the device is releasing resources associated with the data. An example would be deleting a BGP peer, where the peer continues to exist in operational data until the connection is closed and any other resources are released.

To address these issues without requiring a protocol modification, two distinct strategies have been adopted in YANG model design:

The first strategy makes two distinct top-level containers, one for configuration and one for state. These are sometimes referred to as `"/foo"` and `"/foo-state"`. An example would be the interface model defined in [RFC7223]. These models require two completely distinct set of nodes, with repetition of both the interior containers, lists, and key nodes, but also repetition of many other nodes to allow visibility of the operational values of configured nodes. This leads to over-use of YANG groupings in ways that affect the readability of the models, as well as creating opportunities to incorrectly mirror the model's hierarchies. Also this "stitching together" of data from the two trees is merely a convention, not a formal relationship.

The second strategy uses two sibling containers, named `"config"` and `"state"`, placed deeper within the model node hierarchy. The `"config"` container holds the configured values, while the `"state"` container holds the operational values. The duplication of interior nodes in the hierarchies is removed, but the duplication of leafs representing configuration remains. Groupings can be used to avoid the repetition of the leafs in the YANG file, but at the expense of readability. In addition, this strategy does not handle the existence of operational data for which there is no configuration data, such as the system-created data and deleted peers scenarios discussed above.

1.5. Network Management Datastores Architecture

The Network Management Datastores Architecture (NMDA) addresses the problems mentioned above by creating an architectural framework which includes a distinct datastore for operational data, called `<operational>`. This datastore is defined as containing both config true and config false nodes, with the formal understanding that the "in use" values are returned for the config true nodes. This allows modelers to use a single hierarchy for all configuration and operational data, which both improves readability and reduces the possibility of modeling inconsistencies that might impact programmatic access.

In addition, another datastore named `<intended>` is defined to provide a complete view of the configuration data, even in the presence of device-specific features that expand or remove configuration data. While such mechanisms are currently non-standard, the NMDA recognizes they exist and need to be handled appropriately. In the future, such mechanisms may become standardized.

The NMDA allows the deprecation of NETCONF's `<get>` operation, removing the source of these issues. The new operations `<get-data>` and `<edit-data>` will support a parameter indicating the target datastore. Similar changes are planned for RESTCONF ([RFC8040]).

2. Guidelines for YANG Modelers

The following guidelines are meant to help modelers develop YANG models that will maximize the utility of the model with both current implementations and NMDA-capable implementations. Any questions regarding these guidelines can be sent to yang-doctors@ietf.org.

The direction taken should be based on both the maturity of the data model, along with the number of concrete implementations of the model. The intent is not to destabilize the IETF modeling community, but to create models that can take advantage of the NMDA, while pragmatically allowing those models to be used with the existing network configuration protocol implementations.

It is the strong recommendation that models SHOULD move as quickly as possible to the NMDA. This is key to the future of these models. The NETMOD WG will rework existing models to this architecture. Given the permanence and gravity of work published by the IETF, creating future-proof data models is vital.

The two current strategies ("`/foo-state`" and "`config/state`" containers) mix data retrieval details into the data model, complicating the models and impairing their readability. Rather than

maintain these details inside the data model, models can be post-processed to add this derivative information, either manually or using tools.

Tools can automatically produce the required derived modules. The suggested approach is to produce a "state" version of the module with a distinct namespace, rather than using the "/foo-state" top-level container. Since the contents are identical, constraints in the data model such as "must" statements should not need to change. Only the model name, namespace, and prefix should need to change. This simplifies the tooling needed to generate the derived model, as well as reducing changes needed in client applications when transitioning to the NMDA model.

These derived models use distinct module names and namespaces, allowing servers to announce their support for the base or derived models.

Consider the following trivial model:

```
module example-thermostat {
  namespace "tag:ietf:example:thermostat";
  prefix "thermo";

  container thermostat {
    leaf high-temperature {
      description "High temperature threshold";
      type int;
    }
    leaf low-temperature {
      description "Low temperature threshold";
      type int;
    }
    leaf current-temperature {
      description "Current temperature reading";
      type int;
      config false;
    }
  }
}
```

In the derived model, the contents mirror the NMDA data model, but are marked as "config false", and the module name and namespace values have a "-state" suffix:

```
module example-thermostat-state {
  namespace "tag:ietf:example:thermostat-state";
  prefix "thermo-state";

  container thermostat {
    config false;
    leaf high-temperature {
      description "High temperature threshold";
      type int;
    }
    leaf low-temperature {
      description "Low temperature threshold";
      type int;
    }
    leaf current-temperature {
      description "Current temperature reading";
      type int;
    }
  }
}
```

By adopting a tools-based solution for supporting models that are currently under development, models can be quickly restructured to be NMDA-compatible while giving continuity to their community of developers. When NMDA-capable implementations become available, the base data models can be used directly.

Modelers and reviewers can view the simple data model, published in the body of document. Tools can generate any required derived models, and those models can be published in a non-normative appendix to allow interoperability.

It is critical to consider the following guidelines, understanding that our goal is to make models that will see continued use in the long term, balancing short term needs against a desire for consistent, usable models in the future:

(a) New models and models that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA-compatible.

(b) Models that require immediate support for "in use" and "system created" information SHOULD be structured for NMDA. A non-NMDA version of these models SHOULD exist, either an existing model or a model created either by hand or with suitable tools that mirror the current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative

appendix. The use of the non-NMDA model will allow temporary bridging of the time period until NMDA implementations are available.

(c) For published models, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" model in [RFC7223] will be restructured as an NMDA-compatible model. The "/interfaces-state" hierarchy will be marked "status deprecated". Models that mark their "/foo-state" hierarchy with "status deprecated" will allow NMDA-capable implementations to avoid the cost of duplicating the state nodes, while enabling non-NMDA-capable implementations to utilize them for access to the operational values.

(d) For models that augment models which have not been structured with the NMDA, the modeler will have to consider the structure of the base model and the guidelines listed above. Where possible, such models should move to new revisions of the base model that are NMDA-compatible. When that is not possible, augmenting "state" containers SHOULD be avoided, with the expectation that the base model will be re-released with the state containers marked as deprecated. It is RECOMMENDED to augment only the "/foo" hierarchy of the base model. Where this recommendation cannot be followed, then any new "state" elements SHOULD be included in their own module.

3. IANA Considerations

This document has no actions for IANA.

4. Security Considerations

This document has no security considerations.

5. 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-01 (work in progress), March 2017.

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

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

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

Rob Wilton
Cisco Systems

Email: rwilton@cisco.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2017

D. Bogdanovic
Volta Networks
M. Jethanandani
Cisco Systems, Inc
L. Huang
General Electric
S. Agarwal
Cisco Systems, Inc.
D. Blair
Cisco Systems, INC
June 15, 2017

Network Access Control List (ACL) YANG Data Model
draft-ietf-netmod-acl-model-11

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

- o "XXXX" --> the assigned RFC value for this draft.
- o Revision date in model (Oct 12, 2016) 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

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

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

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

This Internet-Draft will expire on December 17, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Definitions and Acronyms	3
2. Problem Statement	4
3. Understanding ACL's Filters and Actions	4
3.1. ACL Modules	5
4. ACL YANG Models	9
4.1. IETF Access Control List module	9
4.2. IETF Packet Fields module	18
4.3. An ACL Example	28
4.4. Port Range Usage Example	28
5. Security Considerations	29
6. IANA Considerations	30
7. Acknowledgements	30
8. Open Issues	31
9. References	31
9.1. Normative References	31
9.2. Informative References	32
Appendix A. Extending ACL model examples	32
A.1. Example of extending existing model for route filtering	32
A.2. A company proprietary module example	34
A.3. Example to augment model with mixed ACL type	42
A.4. Linux nftables	43
Authors' Addresses	44

1. Introduction

Access Control List (ACL) is one of the basic elements to configure device forwarding behavior. It is used in many networking concepts 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 innovations of the networked devices.

Access Control List is also widely known 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 is 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
      +--ro acl-oper-data
      +--rw access-list-entries
        +--rw ace* [rule-name]
          +--rw rule-name      string
          +--rw matches
            | +--rw l2-acl {l2-acl}?
            | | +--rw destination-mac-address?      yang:mac-ad
dress      | |
dress      | | +--rw destination-mac-address-mask? yang:mac-ad
dress      | |
dress      | | +--rw source-mac-address?           yang:mac-ad
dress      | |
dress      | | +--rw source-mac-address-mask?      yang:mac-ad

```

```

| | +--rw ether-type? string
+--rw ipv4-acl {ipv4-acl}?
| | +--rw tos? 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 destination-port-range!
| | +--rw lower-port inet:port-number
| | +--rw upper-port? inet:port-number
+--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 tos? 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 destination-port-range!
| | +--rw lower-port inet:port-number
| | +--rw upper-port? inet:port-number
+--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 ether-type? string

```

		<pre> +--rw tos? 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 destination-port-range! +--rw lower-port inet:port-number +--rw upper-port? inet:port-number +--rw ihl? uint8 +--rw flags? bits +--rw offset? uint16 +--rw identification? uint16 +--rw destination-ipv4-network? inet:ipv4-p </pre>
refix		
		<pre> +--rw source-ipv4-network? inet:ipv4-p </pre>
refix		
		<pre> +--rw l2-l3-ipv6-acl {mixed-ipv6-acl}? +--rw destination-mac-address? yang:mac-ad </pre>
dress		
		<pre> +--rw destination-mac-address-mask? yang:mac-ad </pre>
dress		
		<pre> +--rw source-mac-address? yang:mac-ad </pre>
dress		
		<pre> +--rw source-mac-address-mask? yang:mac-ad </pre>
dress		
		<pre> +--rw ether-type? string +--rw tos? 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 destination-port-range! +--rw lower-port inet:port-number +--rw upper-port? inet:port-number +--rw next-header? uint8 +--rw destination-ipv6-network? inet:ipv6-p </pre>
refix		
		<pre> +--rw source-ipv6-network? inet:ipv6-p </pre>
refix		
		<pre> +--rw flow-label? inet:ipv6-f </pre>
low-label		
		<pre> +--rw l2-l3-ipv4-ipv6-acl {l2-l3-ipv4-ipv6-acl}? +--rw destination-mac-address? yang:mac-ad </pre>
dress		

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 ether-type?	string
			+++rw tos?	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 destination-port-range!	
			+++rw lower-port	inet:port-number
			+++rw upper-port?	inet:port-number
			+++rw ihl?	uint8
			+++rw flags?	bits
			+++rw offset?	uint16
			+++rw identification?	uint16
refix			+++rw destination-ipv4-network?	inet:ipv4-p
refix			+++rw source-ipv4-network?	inet:ipv4-p
refix			+++rw next-header?	uint8
refix			+++rw destination-ipv6-network?	inet:ipv6-p
refix			+++rw source-ipv6-network?	inet:ipv6-p
low-label			+++rw flow-label?	inet:ipv6-f
			+++rw tcp-acl {tcp-acl}?	
			+++rw sequence-number?	uint32
			+++rw acknowledgement-number?	uint32
			+++rw data-offset?	uint8
			+++rw reserved?	uint8
			+++rw flags?	uint16
			+++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 actions
|   +--rw (packet-handling)?
|   |   +--:(deny)
|   |   |   +--rw deny?      empty
|   |   +--:(permit)
|   |   |   +--rw permit?    empty
|   +--rw logging?    boolean
+--ro ace-oper-data
    +--ro match-counter?  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 "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.

<CODE BEGINS> file "ietf-access-control-list@2017-06-16.yang"

```

module ietf-access-control-list {
  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;
  }
  organization
    "IETF NETMOD (NETCONF Data Modeling Language)
     Working Group";

  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
     WG List: netmod@ietf.org
     Editor: Dean Bogdanovic
     ivandean@gmail.com"

```

Editor: Mahesh Jethanandani
mjethanandani@gmail.com
Editor: Lisa Huang
lyihuangl6@gmail.com
Editor: Sonal Agarwal
agarwaso@cisco.com
Editor: Dana Blair
dblair@cisco.com";

description

"This YANG module defines a component that describing the configuration of Access Control Lists (ACLs).
Copyright (c) 2016 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 2017-06-16 {

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.";

}

revision 2016-10-12 {

description

"Base model for Network Access Control List (ACL).";

reference

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

}

/*

* Identities

*/

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.";
}

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

typedef access-control-list-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";
}

/*
 * 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;
      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 acl-oper-data {
    config false;
    description
      "Overall Access Control List operational data";
  }
  container access-list-entries {
    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;
        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 "../.../acl-type = 'eth-acl'";
            uses packet-fields:acl-eth-header-fields;
            description
                "Rule set for L2 ACL.";
        }

        container ipv4-acl {
            if-feature ipv4-acl;
            must "../.../acl-type = '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 "../.../acl-type = '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 "../.../acl-type = '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 "../../../acl-type = '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 "../../../acl-type = '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 "../../../acl-type = 'any-acl'";
    presence "Matches any";
    description
```



```

        "Rule set that allows for a any ACL.";
    }
}

container actions {
    description
        "Definitions of action criteria for this Access List
        Entry.";
    choice packet-handling {
        default "deny";
        description
            "Packet handling action.";
        case deny {
            leaf deny {
                type empty;
                description
                    "Deny action.";
            }
        }
        case permit {
            leaf permit {
                type empty;
                description
                    "Permit action.";
            }
        }
    }
}
leaf logging {
    type boolean;
    description
        "Log the rule on which the match occurred.
        Setting the value to true enables logging,
        whereas setting the value to false disables it.";
}
}
/*
 * Operational state data nodes
 */
container ace-oper-data {
    config false;
    description
        "Operational data for this Access List Entry.";
    leaf match-counter {
        type yang:counter64;
        description
            "Number of matches for this Access List Entry";
    }
}
}

```

```

    }
  }
}

```

<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@2017-06-16.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;
  }

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

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

    Editor: Dean Bogdanovic
    ivandean@gmail.com
    Editor: Mahesh Jethanandani

```

maresh@cisco.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) 2016 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 2017-06-16 {

description

"Added header fields for TCP, UDP, and ICMP.";

reference

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

}

revision 2016-10-12 {

description

"Initial version of packet fields used by ietf-access-control-list";

reference

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

}

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.";
    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.";
    }
}

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.";

    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";
    }
}
}
```

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

  leaf tos {
    type uint8;
    description
      "Also known as Traffic Class in IPv6. The Type of Service (TOS)
       provides an indication of the abstract parameters of the
       quality of service desired.";
    reference
      "RFC 719, RFC 2460";
  }

  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 ether-type {
    type string {
      pattern '[0-9a-fA-F]{4}';
    }
    description
      "The Ethernet Type (or Length) value represented
       in the canonical order defined by IEEE 802.
       The canonical representation uses lowercase
       characters.

       Note: This is not the most ideal way to define
       ether-types. Ether-types are well known types
       and are registered with RAC in IEEE. So they
       should well defined types with values. For now
       this model is defining it as a string."
```



```
        There is a note out to IEEE that needs to be
        turned into a liaison statement asking them to
        define all ether-types for the industry to use.";
    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 uint16;
  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 s 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-type>
        <access-list-entries>
          <ace>
            <rule-name>rule1</rule-name>
            <matches>
              <source-ipv4-network>
                10.10.10.1/24
              </source-ipv4-network>
              <destination-ipv4-network>
                11.11.11.1/24
              </destination-ipv4-network>
            </matches>
            <actions>
              <deny />
            </actions>
            <protocol>
              tcp
            </protocol>
          </ace>
        </access-list-entries>
      </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.

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 draft separately and then work together, to created a new ACL draft that can be supported by different vendors. The new draft removes vendor specific features, and gives examples to allow vendors to extend in their own proprietary ACL. The earlier draft was superseded with the new one that received more participation from many vendors.

Authors would like to thank Jason Sterne, Lada Lhotka, Juergen Schoenwalder, and David Bannister 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.
- o The current model defines 'any' rule as a presence container, allowing a user to define any 'any' rule.
- o The model defines 'ether-type' node as a string. Ideally, this should be a well defined list of all Ethernet Types assigned by IEEE.
- o Should this draft include route-policy definition as defined in draft-ietf-rtgwg-policy-model?

9. References

9.1. Normative References

- [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>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

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-00 (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, <<http://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:access-list-entries/ietf-acl:ace/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
      +--rw v6-upper-bound?    inet:ipv6-prefix

file "example-ext-route-filter@2016-10-12.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 2016-10-12 {
  description
    "Creating Route-Filter extension model based on
    ietf-access-control-list model";
  reference " ";
}
augment "/ietf-acl:access-lists/ietf-acl:acl/"
+ "ietf-acl:access-list-entries/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

Access control list typically does not exist in isolation. Instead, they are associated with a certain scope in which they are applied, for example, an interface of a set of interfaces. How to attach an access control list to an interface (or other system artifact) is outside the scope of this model, as it depends on the specifics of the system model that is being applied. However, in general, the general design pattern will involve adding a data node with a reference, or set of references, to ACLs that are to be applied to the interface. For this purpose, the type definition "access-control-list-ref" can be used.

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, as well how to attach an Access Control List to an interface. 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:access-

list-entries/ 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 input interface or overall packet length. In other example, /ietf-acl:access-lists/ ietf-acl:acl/ietf-acl:access-list-entries/ 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:access-list-entries/ietf-acl:ace/ietf-acl:matches:
  +--rw vlan-tagged?      uint16
  +--rw mpls-unicast?     uint16
  +--rw mpls-multicast?   uint16
  +--rw ipv4?             uint16
  +--rw ipv6?             uint16
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches:
  +--rw ipv4-ttl?         uint8
  +--rw ipv4-len?         uint16
  +--rw ipv4-ihl?         uint8
  +--rw ipv4-id?          uint16
  +--rw ipv4-flags?       ipv4-flags-type
  +--rw ipv4-offset?      uint16
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches:
  +--rw (protocol-payload-choice)?
  |   +---:(protocol-payload)
  |       +--rw protocol-payload* [value-keyword]
  |       +--rw value-keyword      enumeration
  +--rw (metadata)?
  |   +---:(interface-name)
  |       +--rw interface-name* [input-interface]
  |       +--rw input-interface   ietf-if:interface-ref
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/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
augment /ietf-if:interfaces/ietf-if:interface:

```

```

    +--rw acl
      +--rw acl-name?          ietf-acl:access-control-list-ref
      +--ro match-counter?     yang:counter64
      +--rw (direction)?
        +--:(in)
          | +--rw in?          empty
          +--:(out)
            +--rw out?         empty
augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:acl-oper-data:
  +--ro targets
    +--ro (interface)?
      +--:(interface-name)
        +--ro interface-name*  ietf-if:interface-ref

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";
  }

  import ietf-interfaces {
    prefix "ietf-if";
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "Newco model group.";

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

  revision 2016-10-12{
    description
      "Creating NewCo proprietary extensions to ietf-acl model";
    reference
      "RFC XXXX: Network Access Control List (ACL)
       YANG Data Model";
  }
}
```

```
}

typedef known-ether-type {
  type enumeration {
    enum "ipv4" {
      value 2048; // 0x0800
      description "Internet Protocol version 4 (IPv4)";
    }
    enum "vlan-tagged" {
      value 33024; // 0x8100
      description
        "VLAN-tagged frame (IEEE 802.1Q) & Shortest Path
        Bridging IEEE 802.1aq[4]";
    }
    enum "ipv6" {
      value 34525; // 0x86DD
      description "Internet Protocol Version 6 (IPv6)";
    }
    enum "mpls-unicast" {
      value 34887; // 0x8847
      description "MPLS unicast";
    }
    enum "mpls-multicast" {
      value 34888; // 0x8848
      description "MPLS multicast";
    }
  }
  description "Listing supported Ethertypes";
}

typedef ipv4-flags-type {
  type bits {
    bit ipv4-reserved {
      position 0;
      description "reserved bit";
    }
    bit ipv4-DF {
      position 1;
      description "DF bit";
    }
    bit ipv4-MF {
      position 2;
      description "MF bit";
    }
  }
  description "IPv4 flag types";
}
```

```
augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/ietf-acl:ace/" +
    "ietf-acl:matches" {
when "ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:acl-type = 'ace-eth'";

description "additional MAC header matching";

    leaf vlan-tagged {
        type uint16;
        description "Ethernet frame with VLAN tag";
    }

    leaf mpls-unicast {
        type uint16;
        description "Ethernet frame with MPLS unicast payload";
    }

    leaf mpls-multicast {
        type uint16;
        description "Ethernet frame with MPLS multicast payload";
    }

    leaf ipv4 {
        type uint16;
        description "Ethernet frame with IPv4 unicast payload";
    }

    leaf ipv6 {
        type uint16;
        description "Ethernet frame with IPv4 unicast payload";
    }
}
augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/ietf-acl:ace/" +
    "ietf-acl:matches" {
when "ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:acl-type = 'ipv4-acl'";

description "additional IP header information";

    leaf ipv4-ttl {
        type uint8;
        description "time to live of a given packet as
            defined in RFC791";
    }

    leaf ipv4-len {
```

```
    type uint16;
    description "total packet length as defined in RFC791";
}

leaf ipv4-ihl {
    type uint8 {
        range 0..15;
    }
    description "Internet Header Length in 32 bit words
        (see RFC791). Note that while the minimum
        value for this field in a packet is 5,
        we leave open the possibility here that
        the packet has been corrupted.";
}

leaf ipv4-id {
    type uint16;
    description "Identification as decribed in RFC791";
}

leaf ipv4-flags {
    type ipv4-flags-type;
    description "IPv4 flags as defined in RFC791";
}

leaf ipv4-offset {
    type uint16 {
        range 0..8191;
    }
    description "Matches on the packet fragment offset";
}

augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/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";
        list interface-name {
            key input-interface;
            ordered-by user;
            description "Match interface name";
            uses metadata;
        }
    }
}

augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/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 hierarchical-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 metadata {
    description
        "Fields associated with a packet which are not in
        the header.";
    leaf input-interface {
        type ietf-if:interface-ref {
            require-instance false;
        }
        description
            "Packet was received on this interface";
    }
}

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)";
}

augment "/ietf-if:interfaces/ietf-if:interface" {
    description "Apply ACL to interfaces";
    container acl {
        description "ACL related properties.";
        leaf acl-name {
            type ietf-acl:access-control-list-ref;
            description "Access Control List name.";
        }
        leaf match-counter {
            type yang:counter64;
            config false;
        }
    }
}
```

```

        description
            "Total match count for Access Control
            List on this interface";
    }
    choice direction {
        description "Applying ACL in which traffic direction";
        leaf in {
            type empty;
            description "Inbound traffic";
        }
        leaf out {
            type empty;
            description "Outbound traffic";
        }
    }
}

augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:acl-oper-data" {
    description
        "This is an example on how to apply acl to a target to collect
        operational data";
    container targets {
        description "To which object is the ACL attached to";
        choice interface {
            description
                "Access Control List was attached to this interface";
            leaf-list interface-name {
                type ietf-if:interface-ref {
                    require-instance true;
                }
            }
            description "Attached to this interface name";
        }
    }
}
}

```

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. Example to augment model with mixed ACL type

As vendors (or IETF) add more features to ACL, the model is easily augmented. One of such augmentations can be to add support for mixed type of ACLs, where `acl-type-base` can be augmented like in example below:

```
identity mixed-l3-acl {
  base "access-control-list:acl-type-base";
  description "ACL that contains a mix of 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. An acl of type mixed-l3-acl does not contain
    matches on fields in the ethernet header.";
}

identity mixed-l2-l3-acl {
  base "access-control-list:acl-type-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.";
}
```

A.4. 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 firewall 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.

Authors' Addresses

Dean Bogdanovic
Volta Networks

Email: ivandean@gmail.com

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
Cisco Systems, INC

Email: dblair@cisco.com

NETMOD WG
Internet-Draft
Intended status: Standards Track
Expires: May 10, 2019

M. Jethanandani
VMware
S. Agarwal
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

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

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

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

This Internet-Draft will expire on May 10, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Definitions and Acronyms	4
1.2. Terminology	4
1.3. Tree Diagram	4
2. Problem Statement	4
3. Understanding ACL's Filters and Actions	5
3.1. ACL Modules	6
4. ACL YANG Models	10
4.1. IETF Access Control List module	10
4.2. IETF Packet Fields module	24
4.3. ACL Examples	37
4.4. Port Range Usage and Other Examples	39
5. Security Considerations	43
6. IANA Considerations	44
6.1. URI Registration	44
6.2. YANG Module Name Registration	44
7. Acknowledgements	45
8. References	45
8.1. Normative References	45
8.2. Informative References	47
Appendix A. Extending ACL model examples	48
A.1. A company proprietary module example	48
A.2. Linux nftables	51
A.3. Ethertypes	52
Authors' Addresses	60

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 known 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
+---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
+---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
           sagarwall12@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
sagarwall12@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)
            Specification.";
    }
}
```



```
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-ipv6-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 hierarchical-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
    |   +--:(hierarchical-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 firewall 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
             PPoE.";
```

```
}
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 wsmpp {
      value 35036;
      description
        "WAVE Short Message Protocol (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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
July 3, 2017

Common Interface Extension YANG Data Models
draft-ietf-netmod-intf-ext-yang-05

Abstract

This document defines two YANG modules that augment the Interfaces data model defined in the "YANG Data Model for Interface Management" with additional configuration and operational data nodes to support common lower layer interface properties, such as interface MTU. These properties are common to many types of interfaces on network routers and switches and are implemented by multiple network equipment vendors with similar semantics, even though some of the features are not formally defined in any published standard.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.2. Tree Diagrams	4
2. Objectives	4
3. Interfaces Common Module	4
3.1. Reservable Bandwidth	6
3.2. Carrier Delay	6
3.3. Dampening	7
3.3.1. Suppress Threshold	7
3.3.2. Half-Life Period	8
3.3.3. Reuse Threshold	8
3.3.4. Maximum Suppress Time	8
3.4. Encapsulation	8
3.5. Loopback	8
3.6. Layer 2 MTU	9
3.7. Sub-interface	9
3.8. Forwarding Mode	10
4. Interfaces Ethernet-Like Module	10
5. Interfaces Common YANG Module	11
6. Interfaces Ethernet-Like YANG Module	20
7. Open Issues	23
8. Acknowledgements	23
9. ChangeLog	24
9.1. Version -05	24
9.2. Version -04	24
9.3. Version -03	24
9.4. Version -02	24
10. IANA Considerations	24
11. Security Considerations	24
11.1. interfaces-common.yang	25
11.2. interfaces-ethernet-like.yang	26
12. References	26
12.1. Normative References	26
12.2. Informative References	26
Authors' Addresses	27

1. Introduction

This document defines two YANG 1.1 [RFC7950] modules for the management of network interfaces. It defines various augmentations to the generic interfaces data model [RFC7223] to support configuration of lower layer interface properties that are common across many types of network interface.

One of the aims of this draft is to provide a standard namespace and path for these configuration items regardless of the underlying interface type. For example a standard namespace and path for configuring or reading the MAC address associated with an interface is provided that can be used for any interface type that uses Ethernet framing.

Several of the augmentations defined here are not backed by any formal standard specification. Instead, they are for features that are commonly implemented in equivalent ways by multiple independent network equipment vendors. The aim of this draft is to define common paths and leaves for the configuration of these equivalent features in a uniform way, making it easier for users of the YANG model to access these features in a vendor independent way. Where necessary, a description of the expected behavior is also provided with the aim of ensuring vendors implementations are consistent with the specified behaviour.

Given that the modules contain a collection of discrete features with the common theme that they generically apply to interfaces, it is plausible that not all implementors of the YANG module will decide to support all features. Hence separate feature keywords are defined for each logically discrete feature to allow implementors the flexibility to choose which specific parts of the model they support.

The augmentations are split into two separate YANG modules that each focus on a particular area of functionality. The two YANG modules defined in this internet draft are:

ietf-interfaces-common.yang - Defines extensions to the IETF interface data model to support common configuration data nodes.

ietf-interfaces-ethernet-like.yang - Defines a module for any configuration and operational data nodes that are common across interfaces that use Ethernet framing.

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 RFC 2119 [RFC2119].

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration (read-write), and "ro" means state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, 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. Objectives

The aim of the YANG modules contained in this draft is to provide standard definitions for common interface based configuration on network devices.

The expectation is that the YANG leaves that are being defined are fairly widely implemented by network vendors. However, the features described here are mostly not backed by formal standards because they are fairly basic in their behavior and do not need to interoperate with other devices. Where required a concise explanation of the expected behavior is also provided to ensure consistency between vendors.

3. Interfaces Common Module

The Interfaces Common module provides some basic extensions to the IETF interfaces YANG module.

The module provides:

- o A bandwidth configuration leaf to specify the bandwidth available on an interface to control routing metrics.
- o A carrier delay feature used to provide control over short lived link state flaps.
- o An interface link state dampening feature that is used to provide control over longer lived link state flaps.
- o An encapsulation container and extensible choice statement for use by any interface types that allow for configurable L2 encapsulations.
- o A loopback configuration leaf that is primarily aimed at loopback at the physical layer.
- o MTU configuration leaves applicable to all packet/frame based interfaces.
- o A forwarding mode leaf to indicate the OSI layer at which the interface handles traffic
- o A parent interface leaf useable for all types of sub-interface that are children of parent interfaces.

The "ietf-interfaces-common" YANG module has the following structure:

```
module: ietf-interfaces-common
  augment /if:interfaces/if:interface:
    +--rw reservable-bandwidth?  uint64 {reservable-bandwidth}?
    +--rw carrier-delay {carrier-delay}?
    |   +--rw down?  uint32
    |   +--rw up?    uint32
    +--rw dampening! {dampening}?
    |   +--rw half-life?          uint32
    |   +--rw reuse?              uint32
    |   +--rw suppress?           uint32
    |   +--rw max-suppress-time?  uint32
    +--rw encapsulation
    |   +--rw (encaps-type)?
    +--rw loopback?               identityref {loopback}?
    +--rw l2-mtu?                 uint16 {configurable-l2-mtu}?
    +--rw forwarding-mode?        identityref {forwarding-mode}?
  augment /if:interfaces/if:interface:
    +--rw parent-interface        if:interface-ref {sub-interfaces}?
```

3.1. Reservable Bandwidth

The reservable-bandwidth configuration leaf allows the bandwidth of an interface reported to upper layer protocols to be changed (either higher or lower) from the inherent interface bandwidth. The reservable-bandwidth leaf can affect the routing metric cost associated with the interface, but it does not directly limit the amount of traffic that can be sent/received over the interface. If required, interface traffic can be limited to the required bandwidth by configuring an explicit QoS policy.

3.2. Carrier Delay

The carrier delay feature augments the IETF interfaces data model with configuration for a simple algorithm that is used, generally on physical interfaces, to suppress short transient changes in the interface link state. It can be used in conjunction with the dampening feature described in Section 3.3 to provide effective control of unstable links and unwanted state transitions.

The principal of the carrier delay feature is to use a short per interface timer to ensure that any interface link state transition that occurs and reverts back within the specified time interval is entirely suppressed without providing any signalling to any upper layer protocols that the state transition has occurred. E.g. in the case that the link state transition is suppressed then there is no change of the `/if:interfaces-state/if:interface/oper-status` or `/if:interfaces-state/if:interfaces/last-change` leaves for the interface that the feature is operating on. One obvious side effect of using this feature that is worth noting is that any state transition will always be delayed by the specified time interval.

The configuration allows for separate timer values to be used in the suppression of down->up->down link transitions vs up->down->up link transitions.

The carrier delay down timer leaf specifies the amount of time that an interface that is currently in link up state must be continuously down before the down state change is reported to higher level protocols. Use of this timer can cause traffic to be black holed for the configured value and delay reconvergence after link failures, therefore its use is normally restricted to cases where it is necessary to allow enough time for another protection mechanism (such as an optical layer automatic protection system) to take effect.

The carrier delay up timer leaf specifies the amount of time that an interface that is currently in link down state must be continuously up before the down->up link state transition is reported to higher

level protocols. This timer is generally useful as a debounce mechanism to ensure that a link is relatively stable before being brought into service. It can also be used effectively to limit the frequency at which link state transition events can occur. The default value for this leaf is determined by the underlying network device.

3.3. Dampening

The dampening feature introduces a configurable exponential decay mechanism to suppress the effects of excessive interface link state flapping. This feature allows the network operator to configure a device to automatically identify and selectively dampen a local interface which is flapping. Dampening an interface keeps the interface operationally down until the interface stops flapping and becomes stable. Configuring the dampening feature can improve convergence times and stability throughout the network by isolating failures so that disturbances are not propagated, which reduces the utilization of system processing resources by other devices in the network and improves overall network stability.

The basic algorithm uses a counter that is nominally increased by 1000 units every time the underlying interface link state changes from up to down. If the counter increases above the suppress threshold then the interface is kept down (and out of service) until either the maximum suppression time is reached, or the counter has reduced below the reuse threshold. The half-life period determines that rate at which the counter is periodically reduced. Implementations are not required to use a penalty of 1000 units in their dampening algorithm, but should ensure that the Suppress Threshold and Reuse Threshold values are scaled relative to the nominal 1000 unit penalty to ensure that the same configuration values provide consistent behaviour. The configurable values are described in more detail below.

3.3.1. Suppress Threshold

The suppress threshold is the value of the accumulated penalty that triggers the device to dampen a flapping interface. The flapping interface is identified by the device and assigned a penalty for each up to down link state change, but the interface is not automatically dampened. The device tracks the penalties that a flapping interface accumulates. When the accumulated penalty reaches the default or configured suppress threshold, the interface is placed in a dampened state.

3.3.2. Half-Life Period

The half-life period determines how fast the accumulated penalties can decay exponentially. Any penalties that have been accumulated on a flapping interface are reduced by half after each half-life period.

3.3.3. Reuse Threshold

If, after one or more half-life periods, the accumulated penalty decreases below the reuse threshold and the underlying interface link state is up then the interface is taken out of dampened state and allowed to go up.

3.3.4. Maximum Suppress Time

The maximum suppress time represents the maximum amount of time an interface can remain dampened when a penalty is assigned to an interface. The default of the maximum suppress timer is four times the half-life period. The maximum value of the accumulated penalty is calculated using the maximum suppress time, reuse threshold and half-life period.

3.4. Encapsulation

The encapsulation container holds a choice node that is to be augmented with datalink layer specific encapsulations, such as HDLC, PPP, or sub-interface 802.1Q tag match encapsulations. The use of a choice statement ensures that an interface can only have a single datalink layer protocol configured.

The different encapsulations themselves are defined in separate YANG modules defined in other documents that augment the encapsulation choice statement. For example the Ethernet specific basic 'dot1q-vlan' encapsulation is defined in ietf-if-l3-vlan.yang and the 'flexible' encapsulation is defined in ietf-flexible-encapsulation.yang, both modules from [I-D.ietf-netmod-sub-intf-vlan-model].

3.5. Loopback

The loopback configuration leaf allows any physical interface to be configured to be in one of the possible following physical loopback modes, i.e. internal loopback, line loopback, or use of an external loopback connector. The use of YANG identities allows for the model to be extended with other modes of loopback if required.

The following loopback modes are defined:

- o Internal loopback - All egress traffic on the interface is internally looped back within the interface to be received on the ingress path.
- o Line loopback - All ingress traffic received on the interface is internally looped back within the interface to the egress path.
- o Loopback Connector - The interface has a physical loopback connector attached that loops all egress traffic back into the interface's ingress path, with equivalent semantics to internal loopback.

3.6. Layer 2 MTU

A layer 2 MTU configuration leaf (l2-mtu) is provided to specify the maximum size of a layer 2 frame that may be transmitted or received on an interface. The layer 2 MTU includes the overhead of the layer 2 header and the maximum length of the payload, but excludes any frame check sequence (FCS) bytes. The payload MTU available to higher layer protocols is calculated from the l2-mtu leaf after taking the layer 2 header size into account.

For Ethernet interfaces carrying 802.1Q VLAN tagged frames, the l2-mtu excludes the 4-8 byte overhead of any known (e.g. explicitly matched by a child sub-interface) 801.1Q VLAN tags.

3.7. Sub-interface

The sub-interface feature specifies the minimal leaves required to define a child interface that is parented to another interface.

A sub-interface is a logical interface that handles a subset of the traffic on the parent interface. Separate configuration leaves are used to classify the subset of ingress traffic received on the parent interface to be processed in the context of a given sub-interface. All egress traffic processed on a sub-interface is given to the parent interface for transmission. Otherwise, a sub-interface is like any other interface in /if:interfaces and supports the standard interface features and configuration.

For some vendor specific interface naming conventions the name of the child interface is sufficient to determine the parent interface, which implies that the child interface can never be reparented to a different parent interface after it has been created without deleting the existing sub-interface and recreating a new sub-interface. Even in this case it is useful to have a well defined leaf to cleanly identify the parent interface.

The model also allows for arbitrarily named sub-interfaces by having an explicit parent-interface leaf define the child -> parent relationship. In this naming scenario it is also possible for implementations to allow for logical interfaces to be reparented to new parent interfaces without needing the sub-interface to be destroyed and recreated.

3.8. Forwarding Mode

The forwarding mode leaf provides additional information as to what mode or layer an interface is logically operating and forwarding traffic at. The implication of this leaf is that for traffic forwarded at a given layer that any headers for lower layers are stripped off before the packet is forwarded at the given layer. Conversely, on egress any lower layer headers must be added to the packet before it is transmitted out of the interface.

YANG Modules can conditionally use this leaf as a simple mechanism to determine whether particular types of configuration are valid. YANG modules can write 'must' statements to check whether the forwarding mode leaf has been configured, and if it is, then validate that the specified configuration is consistent with any forwarding mode that has also been configured. E.g., a layer 2 QoS policy YANG module could ensure that it is only applied to a interface forwarding traffic at layer 2 by checking whether the forwarding-mode leaf exists, and if it does then also ensure that it has been set to 'layer-2-forwarding'.

The following forwarding modes are defined:

- o Optical Layer - Traffic is being forwarded at the optical layer. This includes DWDM or OTN based switching.
- o Layer 2 - Layer 2 based forwarding, such as Ethernet/VLAN based switching, or L2VPN services.
- o Network Layer - Network layer based forwarding, such as IP, MPLS, or L3VPNs.

4. Interfaces Ethernet-Like Module

The Interfaces Ethernet-Like Module is a small module that contains all configuration and operational data that is common across interface types that use Ethernet framing as their datalink layer encapsulation.

This module currently contains leaves for the configuration and reporting of the operational MAC address and the burnt-in MAC address (BIA) associated with any interface using Ethernet framing.

The "ietf-interfaces-ethernet-like" YANG module has the following structure:

```
module: ietf-interfaces-ethernet-like
  augment /if:interfaces/if:interface:
    +--rw ethernet-like
      +--rw mac-address?      yang:mac-address
      +--ro bia-mac-address?  yang:mac-address
      +--ro statistics
      +--ro in-drop-unknown-dest-mac-pkts? yang:counter64
```

5. Interfaces Common YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223].

```
<CODE BEGINS> file "ietf-interfaces-common@2017-07-03.yang"
module ietf-interfaces-common {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces-common";

  prefix if-cmn;

  import ietf-interfaces {
    prefix if;
  }

  import iana-if-type {
    prefix ianaift;
  }

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

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

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

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

Editor: Robert Wilton
<mailto:rwilton@cisco.com>;

description

"This module contains common definitions for extending the IETF interface YANG model (RFC 7223) with common configurable layer 2 properties.

Copyright (c) 2016, 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 XXX; see the RFC itself for full legal notices.";

```
revision 2017-07-03 {  
  description  
    "Initial version";  
  
  reference "Internet draft: draft-ietf-netmod-intf-ext-yang-05";  
}
```

```
feature reservable-bandwidth {  
  description  
    "This feature indicates that the device supports configuring  
    'reservable-bandwidth' on interfaces.";  
  reference "RFC XXX, Section 3.1 Reservable Bandwidth";  
}
```

```
feature carrier-delay {  
  description  
    "This feature indicates that configurable interface  
    carrier delay is supported, which is a feature is used to  
    limit the propagation of very short interface link state  
    flaps.";  
  reference "RFC XXX, Section 3.2 Carrier Delay";  
}
```

```
feature dampening {
```

```
    description
      "This feature indicates that the device supports interface
      dampening, which is a feature that is used to limit the
      propagation of interface link state flaps over longer
      periods";
    reference "RFC XXX, Section 3.3 Dampening";
  }

  feature loopback {
    description
      "This feature indicates that configurable interface loopback
      is supported.";
    reference "RFC XXX, Section 3.5 Loopback";
  }

  feature configurable-l2-mtu {
    description
      "This feature indicates that the device supports configuring
      layer 2 MTUs on interfaces.  Such MTU configurations include
      the layer 2 header overheads (but exclude any FCS overhead).
      The payload MTU available to higher layer protocols is either
      derived from the layer 2 MTU, taking into account the size of
      the layer 2 header, or is further restricted by explicit layer
      3 or protocol specific MTU configuration.";
    reference "RFC XXX, Section 3.6 Layer 2 MTU";
  }

  feature sub-interfaces {
    description
      "This feature indicates that the device supports the
      instantiation of sub-interfaces.  Sub-interfaces are defined
      as logical child interfaces that allow features and forwarding
      decisions to be applied to a subset of the traffic processed
      on the specified parent interface.";
    reference "RFC XXX, Section 3.7 Sub-interface";
  }

  feature forwarding-mode {
    description
      "This feature indicates that the device supports the
      configurable forwarding mode leaf";
    reference "RFC XXX, Section 3.8 Forwarding Mode";
  }

  /*
   * Define common identities to help allow interface types to be
   * assigned properties.
   */
```

```
identity sub-interface {
  description
    "Base type for generic sub-interfaces.

    New or custom interface types can derive from this type to
    inherit generic sub-interface configuration";
  reference "RFC XXX, Section 3.7 Sub-interface";
}

identity ethSubInterface{
  base ianaift:l2vlan;
  base sub-interface;

  description
    "This identity represents the child sub-interface of any
    interface types that uses Ethernet framing (with or without
    802.1Q tagging)";
}

identity loopback {
  description "Base identity for interface loopback options";
  reference "RFC XXX, section 3.5";
}
identity loopback-internal {
  base loopback;
  description
    "All egress traffic on the interface is internally looped back
    within the interface to be received on the ingress path.";
  reference "RFC XXX, section 3.5";
}
identity loopback-line {
  base loopback;
  description
    "All ingress traffic received on the interface is internally
    looped back within the interface to the egress path.";
  reference "RFC XXX, section 3.5";
}
identity loopback-connector {
  base loopback;
  description
    "The interface has a physical loopback connector attached
    that loops all egress traffic back into the interface's
    ingress path, with equivalent semantics to loopback-internal";
  reference "RFC XXX, section 3.5";
}
```

```
identity forwarding-mode {
  description "Base identity for forwarding-mode options.";
  reference "RFC XXX, section 3.8";
}
identity optical-layer {
  base forwarding-mode;
  description
    "Traffic is being forwarded at the optical layer. This
    includes DWDM or OTN based switching.";
  reference "RFC XXX, section 3.8";
}
identity layer-2-forwarding {
  base forwarding-mode;
  description
    "Layer 2 based forwarding, such as Ethernet/VLAN based
    switching, or L2VPN services.";
  reference "RFC XXX, section 3.8";
}
identity network-layer {
  base forwarding-mode;
  description
    "Network layer based forwarding, such as IP, MPLS, or L3VPNs.";
  reference "RFC XXX, section 3.8";
}

/*
 * Augments the IETF interfaces model with a leaf to explicitly
 * specify the bandwidth available on an interface.
 */
augment "/if:interfaces/if:interface" {
  description
    "Augments the IETF interface model with optional common
    interface level commands that are not formally covered by any
    specific standard.";

  leaf reservable-bandwidth {
    if-feature "reservable-bandwidth";
    type uint64;
    units kbps;
    description
      "The reservable-bandwidth configuration leaf allows the
      bandwidth of an interface reported to upper layer protocols
      to be changed (either higher or lower) from the inherent
      interface bandwidth. The reservable-bandwidth leaf can
      affect the routing metric cost associated with the
      interface, but it does not directly limit the amount of
      traffic that can be sent/received over the interface. If
```

```
        required, interface traffic can be limited to the required
        bandwidth by configuring an explicit QoS policy.";
        reference "RFC XXX, section 3.1";
    }

    /*
     * Defines standard YANG for the Carrier Delay feature.
     */
    container carrier-delay {
        if-feature "carrier-delay";
        description
            "Holds carrier delay related feature configuration";
        leaf down {
            type uint32;
            units milliseconds;
            description
                "Delays the propagation of a 'loss of carrier signal' event
                that would cause the interface state to go down, i.e. the
                command allows short link flaps to be suppressed. The
                configured value indicates the minimum time interval (in
                milliseconds) that the carrier signal must be continuously
                down before the interface state is brought down. If not
                configured, the behaviour on loss of carrier signal is
                vendor/interface specific, but with the general
                expectation that there should be little or no delay.";
        }
        leaf up {
            type uint32;
            units milliseconds;
            description
                "Defines the minimum time interval (in milliseconds) that
                the carrier signal must be continuously present and error
                free before the interface state is allowed to transition
                from down to up. If not configured, the behaviour is
                vendor/interface specific, but with the general
                expectation that sufficient default delay should be used
                to ensure that the interface is stable when enabled before
                being reported as being up. Configured values that are
                too low for the hardware capabilities may be rejected.";
        }
        reference "RFC XXX, Section 3.2 Carrier Delay";
    }

    /*
     * Augments the IETF interfaces model with a container to hold
     * generic interface dampening
     */
    container dampening {
```



```
if-feature "dampening";
presence
  "Enable interface link flap dampening with default settings
  (that are vendor/device specific)";
description
  "Interface dampening limits the propagation of interface link
  state flaps over longer periods";
reference "RFC XXX, Section 3.3 Dampening";
leaf half-life {
  type uint32;
  units seconds;
  description
    "The Time (in seconds) after which a penalty reaches half
    its original value. Once the interface has been assigned
    a penalty, the penalty is decreased by half after the
    half-life period. For some devices, the allowed values may
    be restricted to particular multiples of seconds. The
    default value is vendor/device specific.";
  reference "RFC XXX, Section 3.3.2 Half-Life Period";
}
leaf reuse {
  type uint32;
  description
    "Penalty value below which a stable interface is
    unsuppressed (i.e. brought up) (no units). The default
    value is vendor/device specific. The penalty value for a
    link up->down state change is nominally 1000 units.";
  reference "RFC XXX, Section 3.3.3 Reuse Threshold";
}

leaf suppress {
  type uint32;
  description
    "Limit at which an interface is suppressed (i.e. held down)
    when its penalty exceeds that limit (no units). The value
    must be greater than the reuse threshold. The default
    value is vendor/device specific. The penalty value for a
    link up->down state change is nominally 1000 units.";
  reference "RFC XXX, Section 3.3.1 Suppress Threshold";
}

leaf max-suppress-time {
  type uint32;
  units seconds;
  description
    "Maximum time (in seconds) that an interface can be
    suppressed. This value effectively acts as a ceiling that
    the penalty value cannot exceed. The default value is
```

```
        vendor/device specific.";
    reference "RFC XXX, Section 3.3.4 Maximum Suppress Time";
}

/*
 * Various types of interfaces support a configurable layer 2
 * encapsulation, any that are supported by YANG should be
 * listed here.
 *
 * Different encapsulations can hook into the common encaps-type
 * choice statement.
 */
container encapsulation {
    when
        "derived-from-or-self(..if:type,
                                'ianaift:ethernetCsmacd') or
        derived-from-or-self(..if:type,
                                'ianaift:ieee8023adLag') or
        derived-from-or-self(..if:type, 'ianaift:pos') or
        derived-from-or-self(..if:type,
                                'ianaift:atmSubInterface') or
        derived-from-or-self(..if:type, 'ethSubInterface')" {

        description
            "All interface types that can have a configurable L2
            encapsulation";
    }

    description
        "Holds the OSI layer 2 encapsulation associated with an
        interface";
    choice encaps-type {
        description
            "Extensible choice of layer 2 encapsulations";
        reference "RFC XXX, Section 3.4 Encapsulation";
    }
}

/*
 * Various types of interfaces support loopback configuration,
 * any that are supported by YANG should be listed here.
 */
leaf loopback {
    when "derived-from-or-self(..if:type,
                                'ianaift:ethernetCsmacd') or
        derived-from-or-self(..if:type, 'ianaift:sonet') or
        derived-from-or-self(..if:type, 'ianaift:atm') or
```

```
        derived-from-or-self(..if:type, 'ianaift:otnOtu')" {
        description
            "All interface types that support loopback configuration.";
        }
        if-feature "loopback";
        type identityref {
            base loopback;
        }
        description "Enables traffic loopback.";
        reference "RFC XXX, Section 3.5 Loopback";
    }

    /*
     * Many types of interfaces support a configurable layer 2 MTU.
     */
    leaf l2-mtu {
        if-feature "configurable-l2-mtu";
        type uint16 {
            range "64 .. 65535";
        }
        description
            "The maximum size of layer 2 frames that may be transmitted
             or received on the interface (excluding any FCS overhead).
             In the case of Ethernet interfaces it also excludes the
             4-8 byte overhead of any known (i.e. explicitly matched by
             a child sub-interface) 801.1Q VLAN tags.";
        reference "RFC XXX, Section 3.6 Layer 2 MTU";
    }

    /*
     * Augments the IETF interfaces model with a leaf that indicates
     * which mode, or layer, is being used to forward the traffic.
     */
    leaf forwarding-mode {
        if-feature "forwarding-mode";
        type identityref {
            base forwarding-mode;
        }

        description
            "The forwarding mode that the interface is operating in.";
        reference "RFC XXX, Section 3.8 Forwarding Mode";
    }
}

/*
 * Add generic support for sub-interfaces.
 */
```

```

* This should be extended to cover all interface types that are
* child interfaces of other interfaces.
*/
augment "/if:interfaces/if:interface" {
  when "derived-from(if:type, 'sub-interface') or
        derived-from-or-self(if:type, 'ianaift:atmSubInterface') or
        derived-from-or-self(if:type, 'ianaift:frameRelay')" {
    description
      "Any ianaift:types that explicitly represent sub-interfaces
      or any types that derive from the sub-interface identity";
  }
  if-feature "sub-interfaces";

  description
    "Add a parent interface field to interfaces that model
    sub-interfaces";
  leaf parent-interface {

    type if:interface-ref;

    mandatory true;
    description
      "This is the reference to the parent interface of this
      sub-interface.";
    reference "RFC XXX, Section 3.7 Sub-interface";
  }
}
}
}
<CODE ENDS>

```

6. Interfaces Ethernet-Like YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223] for Ethernet-like interfaces. This includes Ethernet interfaces, 802.3 LAG (802.1AX) interfaces, VLAN sub-interfaces, Switch Virtual interfaces, and Pseudo-Wire Head-End interfaces.

```

<CODE BEGINS> file "ietf-interfaces-ethernet-like@2017-07-03.yang"
module ietf-interfaces-ethernet-like {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-interfaces-ethernet-like";

  prefix ethlike;

```

```
import ietf-interfaces {
  prefix if;
}

import ietf-yang-types {
  prefix yang;
}

import iana-if-type {
  prefix ianaift;
}

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

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

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

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

  Editor:     Robert Wilton
              <mailto:rwilton@cisco.com>";

description
  "This module contains YANG definitions for configuration for
  'Ethernet-like' interfaces.  It is applicable to all interface
  types that use Ethernet framing and expose an Ethernet MAC
  layer, and includes such interfaces as physical Ethernet
  interfaces, Ethernet LAG interfaces and VLAN sub-interfaces.

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

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

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

```
revision 2017-07-03 {
  description "Updated to conform to NMDA architecture";

  reference
    "Internet draft: draft-ietf-netmod-intf-ext-yang-05";
}

/*
 * Configuration parameters for Ethernet-like interfaces.
 */
augment "/if:interfaces/if:interface" {
  when "derived-from-or-self(if:type, 'ianaift:ethernetCsmacd') or
        derived-from-or-self(if:type, 'ianaift:ieee8023adLag') or
        derived-from-or-self(if:type, 'ianaift:l2vlan') or
        derived-from-or-self(if:type, 'ianaift:ifPwType')" {
    description "Applies to all Ethernet-like interfaces";
  }
  description
    "Augment the interface model with parameters for all
    Ethernet-like interfaces";

  container ethernet-like {
    description
      "Contains parameters for interfaces that use Ethernet framing
      and expose an Ethernet MAC layer.";
    leaf mac-address {
      type yang:mac-address;
      description
        "The MAC address of the interface.";
    }

    leaf bia-mac-address {
      type yang:mac-address;
      config false;
      description
        "The 'burnt-in' MAC address. I.e the default MAC address
        assigned to the interface if no MAC address has been
        explicitly configured on it.";
    }
  }

  container statistics {
    config false;
    description
      "Packet statistics that apply to all Ethernet-like
      interfaces";
    leaf in-drop-unknown-dest-mac-pkts {
      type yang:counter64;
      units frames;
    }
  }
}
```

`description`

"A count of the number of frames that were well formed, but otherwise dropped because the destination MAC address did not pass any ingress destination MAC address filter.

For consistency, frames counted against this drop counters are also counted against the IETF interfaces statistics. In particular, they are included in in-octets and in-discards, but are not included in in-unicast-pkts, in-multicast-pkts or in-broadcast-pkts, because they are not delivered to a higher layer.

Discontinuities in the values of this counters in this container can occur at re-initialization of the management system, and at other times as indicated by the value of the 'discontinuity-time' leaf defined in the ietf-interfaces YANG module (RFC 7223).";

```
    }  
  }  
}  
}  
}  
<CODE ENDS>
```

7. Open Issues

Open issues:

1. Consider whether to use interface property identities (as per draft-wilton-netmod-interface-properties).
2. Provide configuration examples?
3. Provide -state module for Ethernet-like

8. Acknowledgements

The authors wish to thank Eric Gray, Ing-Wher Chen, Juergen Schoenwaelder, Ladislav Lhotka, Mahesh Jethanandani, Michael Zitao, Neil Ketley, Qin Wu, William Lupton, Xufeng Liu, and Andy Bierman for their helpful comments contributing to this draft.

9. ChangeLog

9.1. Version -05

- o Incorporate feedback from Andy Bierman

9.2. Version -04

- o Incorporate feedback from Lada, some comments left as open issues.

9.3. Version -03

- o Fixed incorrect module name references, and updated tree output

9.4. Version -02

- o Minor changes only: Fix errors in when statements, use derived-from-or-self() for future proofing.

10. IANA Considerations

This document defines several new YANG module and the authors politely request that IANA assigns unique names to the YANG module files contained within this draft, and also appropriate URIs in the "IETF XML Registry".

11. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [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 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:

11.1.1. interfaces-common.yang

The interfaces-common YANG module contains various configuration leaves that affect the behavior of interfaces. Modifying these leaves can cause an interface to go down, or become unreliable, or to drop traffic forwarded over it. More specific details of the possible failure modes are given below.

The following leaf could cause the interface to go down, and stop processing any ingress or egress traffic on the interface:

- o /if:interfaces/if:interface/loopback

The following leaf could cause changes to the routing metrics. Any change in routing metrics could cause too much traffic to be routed through the interface, or through other interfaces in the network, potentially causing traffic loss due to excessive traffic on a particular interface or network device:

- o /if:interfaces/if:interface/bandwidth

The following leaves could cause instabilities at the interface link layer, and cause unwanted higher layer routing path changes if the leaves are modified, although they would generally only affect a device that had some underlying link stability issues:

- o /if:interfaces/if:interface/carrier-delay/down

- o /if:interfaces/if:interface/carrier-delay/up

- o /if:interfaces/if:interface/dampening/half-life

- o /if:interfaces/if:interface/dampening/reuse

- o /if:interfaces/if:interface/dampening/suppress

- o /if:interfaces/if:interface/dampening/max-suppress-time

The following leaves could cause traffic loss on the interface because the received or transmitted frames do not comply with the frame matching criteria on the interface and hence would be dropped:

- o /if:interfaces/if:interface/encapsulation

- o /if:interfaces/if:interface/l2-mtu

- o /if:interfaces/if:interface/forwarding-mode

Normally devices will not allow the parent-interface leaf to be changed after the interface has been created. If an implementation did allow the parent-interface leaf to be changed then it could cause all traffic on the affected interface to be dropped. The affected leaf is:

- o /if:interfaces/if:interface/parent-interface

11.2. interfaces-ethernet-like.yang

Generally, the configuration nodes in the interfaces-ethernet-like YANG module are concerned with configuration that is common across all types of Ethernet-like interfaces. Currently, the module only contains a node for configuring the operational MAC address to use on an interface. Adding/modifying/deleting this leaf has the potential risk of causing protocol instability, excessive protocol traffic, and general traffic loss, particularly if the configuration change caused a duplicate MAC address to be present on the local network. The following leaf is affected:

- o interfaces/interface/ethernet-like/mac-address

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>>.
- [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>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<http://www.rfc-editor.org/info/rfc7224>>.
- [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.ietf-netmod-sub-intf-vlan-model]

Wilton, R., Ball, D., tapsingh@cisco.com, t., and S. Sivaraj, "Sub-interface VLAN YANG Data Models", draft-ietf-netmod-sub-intf-vlan-model-01 (work in progress), March 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>>.

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

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

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@juniper.net

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 30, 2021

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
July 29, 2020

Common Interface Extension YANG Data Models
draft-ietf-netmod-intf-ext-yang-10

Abstract

This document defines two YANG modules that augment the Interfaces data model defined in the "YANG Data Model for Interface Management" with additional configuration and operational data nodes to support common lower layer interface properties, such as interface MTU.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) defined in RFC 8342.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 30, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.2. Tree Diagrams	4
2. Interface Extensions Module	4
2.1. Carrier Delay	5
2.2. Dampening	6
2.2.1. Suppress Threshold	7
2.2.2. Half-Life Period	7
2.2.3. Reuse Threshold	7
2.2.4. Maximum Suppress Time	7
2.3. Encapsulation	7
2.4. Loopback	8
2.5. Maximum frame size	8
2.6. Sub-interface	8
2.7. Forwarding Mode	9
3. Interfaces Ethernet-Like Module	9
4. Interface Extensions YANG Module	10
5. Interfaces Ethernet-Like YANG Module	21
6. Examples	25
6.1. Carrier delay configuration	25
6.2. Dampening configuration	26
6.3. MAC address configuration	27
7. Acknowledgements	29
8. ChangeLog	29
8.1. Version -10	29
8.2. Version -09	29
8.3. Version -08	29
8.4. Version -07	29
8.5. Version -06	29
8.6. Version -05	29
8.7. Version -04	29
8.8. Version -03	30
8.9. Version -02	30
9. IANA Considerations	30
9.1. YANG Module Registrations	30
10. Security Considerations	31
10.1. ietf-if-extensions.yang	31
10.2. ietf-if-ethernet-like.yang	32
11. References	32
11.1. Normative References	32

11.2. Informative References	33
Authors' Addresses	34

1. Introduction

This document defines two NMDA compatible [RFC8342] YANG 1.1 [RFC7950] modules for the management of network interfaces. It defines various augmentations to the generic interfaces data model [RFC8343] to support configuration of lower layer interface properties that are common across many types of network interface.

One of the aims of this document is to provide a standard definition for these configuration items regardless of the underlying interface type. For example, a definition for configuring or reading the MAC address associated with an interface is provided that can be used for any interface type that uses Ethernet framing.

Several of the augmentations defined here are not backed by any formal standard specification. Instead, they are for features that are commonly implemented in equivalent ways by multiple independent network equipment vendors. The aim of this document is to define common paths and leaves for the configuration of these equivalent features in a uniform way, making it easier for users of the YANG model to access these features in a vendor independent way. Where necessary, a description of the expected behavior is also provided with the aim of ensuring vendors implementations are consistent with the specified behaviour.

Given that the modules contain a collection of discrete features with the common theme that they generically apply to interfaces, it is plausible that not all implementors of the YANG module will decide to support all features. Hence separate feature keywords are defined for each logically discrete feature to allow implementors the flexibility to choose which specific parts of the model they support.

The augmentations are split into two separate YANG modules that each focus on a particular area of functionality. The two YANG modules defined in this document are:

`ietf-if-extensions.yang` - Defines extensions to the IETF interface data model to support common configuration data nodes.

`ietf-if-ethernet-like.yang` - Defines a module for any configuration and operational data nodes that are common across interfaces that use Ethernet framing.

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 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Tree Diagrams

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

2. Interface Extensions Module

The Interfaces Extensions YANG module provides some basic extensions to the IETF interfaces YANG module.

The module provides:

- o A carrier delay feature used to provide control over short lived link state flaps.
- o An interface link state dampening feature that is used to provide control over longer lived link state flaps.
- o An encapsulation container and extensible choice statement for use by any interface types that allow for configurable L2 encapsulations.
- o A loopback configuration leaf that is primarily aimed at loopback at the physical layer.
- o MTU configuration leaves applicable to all packet/frame based interfaces.
- o A forwarding mode leaf to indicate the OSI layer at which the interface handles traffic.
- o A generic "sub-interface" identity that an interface identity definition can derive from if it defines a sub-interface.
- o A parent interface leaf useable for all types of sub-interface that are children of parent interfaces.

The "ietf-if-extensions" YANG module has the following structure:

```

module: ietf-if-extensions
  augment /if:interfaces/if:interface:
    +--rw carrier-delay {carrier-delay}?
    |   +--rw down?                uint32
    |   +--rw up?                  uint32
    |   +--ro carrier-transitions? yang:counter64
    |   +--ro timer-running?       enumeration
    +--rw dampening! {dampening}?
    |   +--rw half-life?           uint32
    |   +--rw reuse?               uint32
    |   +--rw suppress?            uint32
    |   +--rw max-suppress-time?   uint32
    |   +--ro penalty?             uint32
    |   +--ro suppressed?          boolean
    |   +--ro time-remaining?      uint32
    +--rw encapsulation
    |   +--rw (encaps-type)?
    +--rw loopback?               identityref {loopback}?
    +--rw max-frame-size?         uint32 {max-frame-size}?
    +--ro forwarding-mode?        identityref
  augment /if:interfaces/if:interface:
    +--rw parent-interface        if:interface-ref {sub-interfaces}?
  augment /if:interfaces/if:interface/if:statistics:
    +--ro in-discard-unknown-encaps? yang:counter64
        {sub-interfaces}?

```

2.1. Carrier Delay

The carrier delay feature augments the IETF interfaces data model with configuration for a simple algorithm that is used, generally on physical interfaces, to suppress short transient changes in the interface link state. It can be used in conjunction with the dampening feature described in Section 2.2 to provide effective control of unstable links and unwanted state transitions.

The principle of the carrier delay feature is to use a short per interface timer to ensure that any interface link state transition that occurs and reverts back within the specified time interval is entirely suppressed without providing any signalling to any upper layer protocols that the state transition has occurred. E.g. in the case that the link state transition is suppressed then there is no change of the /if:interfaces/if:interface/oper-status or /if:interfaces/if:interfaces/last-change leaves for the interface that the feature is operating on. One obvious side effect of using

this feature that is that any state transition will always be delayed by the specified time interval.

The configuration allows for separate timer values to be used in the suppression of down->up->down link transitions vs up->down->up link transitions.

The carrier delay down timer leaf specifies the amount of time that an interface that is currently in link up state must be continuously down before the down state change is reported to higher level protocols. Use of this timer can cause traffic to be black holed for the configured value and delay reconvergence after link failures, therefore its use is normally restricted to cases where it is necessary to allow enough time for another protection mechanism (such as an optical layer automatic protection system) to take effect.

The carrier delay up timer leaf specifies the amount of time that an interface that is currently in link down state must be continuously up before the down->up link state transition is reported to higher level protocols. This timer is generally useful as a debounce mechanism to ensure that a link is relatively stable before being brought into service. It can also be used effectively to limit the frequency at which link state transition events may occur. The default value for this leaf is determined by the underlying network device.

2.2. Dampening

The dampening feature introduces a configurable exponential decay mechanism to suppress the effects of excessive interface link state flapping. This feature allows the network operator to configure a device to automatically identify and selectively dampen a local interface which is flapping. Dampening an interface keeps the interface operationally down until the interface stops flapping and becomes stable. Configuring the dampening feature can improve convergence times and stability throughout the network by isolating failures so that disturbances are not propagated, which reduces the utilization of system processing resources by other devices in the network and improves overall network stability.

The basic algorithm uses a counter that is increased by 1000 units every time the underlying interface link state changes from up to down. If the counter increases above the suppress threshold then the interface is kept down (and out of service) until either the maximum suppression time is reached, or the counter has reduced below the reuse threshold. The half-life period determines that rate at which the counter is periodically reduced by half.

2.2.1. Suppress Threshold

The suppress threshold is the value of the accumulated penalty that triggers the device to dampen a flapping interface. The flapping interface is identified by the device and assigned a penalty for each up to down link state change, but the interface is not automatically dampened. The device tracks the penalties that a flapping interface accumulates. When the accumulated penalty reaches or exceeds the suppress threshold, the interface is placed in a suppressed state.

2.2.2. Half-Life Period

The half-life period determines how fast the accumulated penalties can decay exponentially. The accumulated penalty decays at a rate that causes its value to be reduced by half after each half-life period.

2.2.3. Reuse Threshold

If, after one or more half-life periods, the accumulated penalty decreases below the reuse threshold and the underlying interface link state is up then the interface is taken out of suppressed state and is allowed to go up.

2.2.4. Maximum Suppress Time

The maximum suppress time represents the maximum amount of time an interface can remain dampened when a new penalty is assigned to an interface. The default of the maximum suppress timer is four times the half-life period. The maximum value of the accumulated penalty is calculated using the maximum suppress time, reuse threshold and half-life period.

2.3. Encapsulation

The encapsulation container holds a choice node that is to be augmented with datalink layer specific encapsulations, such as HDLC, PPP, or sub-interface 802.1Q tag match encapsulations. The use of a choice statement ensures that an interface can only have a single datalink layer protocol configured.

The different encapsulations themselves are defined in separate YANG modules defined in other documents that augment the encapsulation choice statement. For example the Ethernet specific basic 'dot1q-vlan' encapsulation is defined in ietf-if-l3-vlan.yang and the 'flexible' encapsulation is defined in ietf-flexible-encapsulation.yang, both modules from [I-D.ietf-netmod-sub-intf-vlan-model].

2.4. Loopback

The loopback configuration leaf allows any physical interface to be configured to be in one of the possible following physical loopback modes, i.e. internal loopback, line loopback, or use of an external loopback connector. The use of YANG identities allows for the model to be extended with other modes of loopback if required.

The following loopback modes are defined:

- o Internal loopback - All egress traffic on the interface is internally looped back within the interface to be received on the ingress path.
- o Line loopback - All ingress traffic received on the interface is internally looped back within the interface to the egress path.
- o Loopback Connector - The interface has a physical loopback connector attached that loops all egress traffic back into the interface's ingress path, with equivalent semantics to internal loopback.

2.5. Maximum frame size

A maximum frame size configuration leaf (max-frame-size) is provided to specify the maximum size of a layer 2 frame that may be transmitted or received on an interface. The value includes the overhead of any layer 2 header, the maximum length of the payload, and any frame check sequence (FCS) bytes. If configured, the max-frame-size leaf on an interface also restricts the max-frame-size of any child sub-interfaces, and the available MTU for protocols.

2.6. Sub-interface

The sub-interface feature specifies the minimal leaves required to define a child interface that is parented to another interface.

A sub-interface is a logical interface that handles a subset of the traffic on the parent interface. Separate configuration leaves are used to classify the subset of ingress traffic received on the parent interface to be processed in the context of a given sub-interface. All egress traffic processed on a sub-interface is given to the parent interface for transmission. Otherwise, a sub-interface is like any other interface in /if:interfaces and supports the standard interface features and configuration.

For some vendor specific interface naming conventions the name of the child interface is sufficient to determine the parent interface,

which implies that the child interface can never be reparented to a different parent interface after it has been created without deleting the existing sub-interface and recreating a new sub-interface. Even in this case it is useful to have a well defined leaf to cleanly identify the parent interface.

The model also allows for arbitrarily named sub-interfaces by having an explicit parent-interface leaf define the child -> parent relationship. In this naming scenario it is also possible for implementations to allow for logical interfaces to be reparented to new parent interfaces without needing the sub-interface to be destroyed and recreated.

2.7. Forwarding Mode

The forwarding mode leaf provides additional information as to what mode or layer an interface is logically operating and forwarding traffic at. The implication of this leaf is that for traffic forwarded at a given layer that any headers for lower layers are stripped off before the packet is forwarded at the given layer. Conversely, on egress any lower layer headers must be added to the packet before it is transmitted out of the interface.

The following forwarding modes are defined:

- o Physical - Traffic is being forwarded at the physical layer. This includes DWDM or OTN based switching.
- o Data-link - Layer 2 based forwarding, such as Ethernet/VLAN based switching, or L2VPN services.
- o Network - Network layer based forwarding, such as IP, MPLS, or L3VPNs.

3. Interfaces Ethernet-Like Module

The Interfaces Ethernet-Like Module is a small module that contains all configuration and operational data that is common across interface types that use Ethernet framing as their datalink layer encapsulation.

This module currently contains leaves for the configuration and reporting of the operational MAC address and the burnt-in MAC address (BIA) associated with any interface using Ethernet framing.

The "ietf-if-ethernet-like" YANG module has the following structure:

```
module: ietf-if-ethernet-like
  augment /if:interfaces/if:interface:
    +--rw ethernet-like
      +--rw mac-address?      yang:mac-address
      |      {configurable-mac-address}?
      +--ro bia-mac-address?  yang:mac-address
  augment /if:interfaces/if:interface/if:statistics:
    +--ro in-drop-unknown-dest-mac-pkts?  yang:counter64
```

4. Interface Extensions YANG Module

This YANG module augments the interface container defined in [RFC8343]. It also contains references to [RFC6991] and [RFC7224].

```
<CODE BEGINS> file "ietf-if-extensions@2020-07-29.yang"
module ietf-if-extensions {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-if-extensions";

  prefix if-ext;

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

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

  import iana-if-type {
    prefix ianaift;
    reference "RFC 7224: IANA Interface Type YANG Module";
  }

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

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

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

Editor: Robert Wilton
<mailto:rwilton@cisco.com>;

description

"This module contains common definitions for extending the IETF interface YANG model (RFC 8343) with common configurable layer 2 properties.

Copyright (c) 2020 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>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

revision 2020-07-29 {

description

"Initial revision.";

reference

"RFC XXXX, Common Interface Extension YANG Data Models";

}

feature carrier-delay {

description

"This feature indicates that configurable interface carrier delay is supported, which is a feature is used to limit the propagation of very short interface link state flaps.";

reference "RFC XXXX, Section 2.1 Carrier Delay";

}

feature dampening {

description

```
        "This feature indicates that the device supports interface
        dampening, which is a feature that is used to limit the
        propagation of interface link state flaps over longer
        periods.";
    reference "RFC XXXX, Section 2.2 Dampening";
}

feature loopback {
    description
        "This feature indicates that configurable interface loopback is
        supported.";
    reference "RFC XXXX, Section 2.4 Loopback";
}

feature max-frame-size {
    description
        "This feature indicates that the device supports configuring or
        reporting the maximum frame size on interfaces.";
    reference "RFC XXXX, Section 2.5 Maximum Frame Size";
}

feature sub-interfaces {
    description
        "This feature indicates that the device supports the
        instantiation of sub-interfaces. Sub-interfaces are defined
        as logical child interfaces that allow features and forwarding
        decisions to be applied to a subset of the traffic processed
        on the specified parent interface.";
    reference "RFC XXXX, Section 2.6 Sub-interface";
}

/*
 * Define common identities to help allow interface types to be
 * assigned properties.
 */
identity sub-interface {
    description
        "Base type for generic sub-interfaces.

        New or custom interface types can derive from this type to
        inherit generic sub-interface configuration.";
    reference "RFC XXXX, Section 2.6 Sub-interface";
}

identity ethSubInterface{
    base ianaift:l2vlan;
    base sub-interface;
```

```
    description
      "This identity represents the child sub-interface of any
       interface types that uses Ethernet framing (with or without
       802.1Q tagging).";
  }

  identity loopback {
    description "Base identity for interface loopback options";
    reference "RFC XXXX, Section 2.4";
  }
  identity internal {
    base loopback;
    description
      "All egress traffic on the interface is internally looped back
       within the interface to be received on the ingress path.";
    reference "RFC XXXX, Section 2.4";
  }
  identity line {
    base loopback;
    description
      "All ingress traffic received on the interface is internally
       looped back within the interface to the egress path.";
    reference "RFC XXXX, Section 2.4";
  }
  identity connector {
    base loopback;
    description
      "The interface has a physical loopback connector attached that
       loops all egress traffic back into the interface's ingress
       path, with equivalent semantics to loopback internal.";
    reference "RFC XXXX, Section 2.4";
  }

  identity forwarding-mode {
    description "Base identity for forwarding-mode options.";
    reference "RFC XXXX, Section 2.7";
  }
  identity physical {
    base forwarding-mode;
    description
      "Physical layer forwarding. This includes DWDM or OTN based
       optical switching.";
    reference "RFC XXXX, Section 2.7";
  }
  identity data-link {
    base forwarding-mode;
    description
```



```
        "Layer 2 based forwarding, such as Ethernet/VLAN based
        switching, or L2VPN services.";
    reference "RFC XXXX, Section 2.7";
}
identity network {
    base forwarding-mode;
    description
        "Network layer based forwarding, such as IP, MPLS, or L3VPNs.";
    reference "RFC XXXX, Section 2.7";
}

/*
 * Augments the IETF interfaces model with leaves to configure
 * and monitor carrier-delay on an interface.
 */
augment "/if:interfaces/if:interface" {
    description
        "Augments the IETF interface model with optional common
        interface level commands that are not formally covered by any
        specific standard.";

    /*
     * Defines standard YANG for the Carrier Delay feature.
     */
    container carrier-delay {
        if-feature "carrier-delay";
        description
            "Holds carrier delay related feature configuration.";
        leaf down {
            type uint32;
            units milliseconds;
            description
                "Delays the propagation of a 'loss of carrier signal' event
                that would cause the interface state to go down, i.e. the
                command allows short link flaps to be suppressed. The
                configured value indicates the minimum time interval (in
                milliseconds) that the carrier signal must be continuously
                down before the interface state is brought down. If not
                configured, the behaviour on loss of carrier signal is
                vendor/interface specific, but with the general
                expectation that there should be little or no delay.";
        }
        leaf up {
            type uint32;
            units milliseconds;
            description
                "Defines the minimum time interval (in milliseconds) that
```

```
the carrier signal must be continuously present and error
free before the interface state is allowed to transition
from down to up.  If not configured, the behaviour is
vendor/interface specific, but with the general
expectation that sufficient default delay should be used
to ensure that the interface is stable when enabled before
being reported as being up.  Configured values that are
too low for the hardware capabilities may be rejected.";
}
leaf carrier-transitions {
  type yang:counter64;
  units transitions;
  config false;
  description
    "Defines the number of times the underlying carrier state
    has changed to, or from, state up.  This counter should be
    incremented even if the high layer interface state changes
    are being suppressed by a running carrier-delay timer.";
}
leaf timer-running {
  type enumeration {
    enum none {
      description
        "No carrier delay timer is running.";
    }
    enum up {
      description
        "Carrier-delay up timer is running.  The underlying
        carrier state is up, but interface state is not
        reported as up.";
    }
    enum down {
      description
        "Carrier-delay down timer is running.  Interface state
        is reported as up, but the underlying carrier state is
        actually down.";
    }
  }
  config false;
  description
    "Reports whether a carrier delay timer is actively running,
    in which case the interface state does not match the
    underlying carrier state.";
}

reference "RFC XXXX, Section 2.1 Carrier Delay";
}
```

```
/*
 * Augments the IETF interfaces model with a container to hold
 * generic interface dampening
 */
container dampening {
  if-feature "dampening";
  presence
    "Enable interface link flap dampening with default settings
    (that are vendor/device specific).";
  description
    "Interface dampening limits the propagation of interface link
    state flaps over longer periods.";
  reference "RFC XXXX, Section 2.2 Dampening";

  leaf half-life {
    type uint32;
    units seconds;
    description
      "The time (in seconds) after which a penalty would be half
      its original value. Once the interface has been assigned
      a penalty, the penalty is decreased at a decay rate
      equivalent to the half-life. For some devices, the
      allowed values may be restricted to particular multiples
      of seconds. The default value is vendor/device
      specific.";
    reference "RFC XXXX, Section 2.3.2 Half-Life Period";
  }

  leaf reuse {
    type uint32;
    description
      "Penalty value below which a stable interface is
      unsuppressed (i.e. brought up) (no units). The default
      value is vendor/device specific. The penalty value for a
      link up->down state change is 1000 units.";
    reference "RFC XXXX, Section 2.2.3 Reuse Threshold";
  }

  leaf suppress {
    type uint32;
    description
      "Limit at which an interface is suppressed (i.e. held down)
      when its penalty exceeds that limit (no units). The value
      must be greater than the reuse threshold. The default
      value is vendor/device specific. The penalty value for a
      link up->down state change is 1000 units.";
    reference "RFC XXXX, Section 2.2.1 Suppress Threshold";
  }
}
```

```
leaf max-suppress-time {
    type uint32;
    units seconds;
    description
        "Maximum time (in seconds) that an interface can be
        suppressed before being unsuppressed if no further link
        up->down state change penalties have been applied. This
        value effectively acts as a ceiling that the penalty value
        cannot exceed. The default value is vendor/device
        specific.";
    reference "RFC XXXX, Section 2.2.4 Maximum Suppress Time";
}

leaf penalty {
    type uint32;
    config false;
    description
        "The current penalty value for this interface. When the
        penalty value exceeds the 'suppress' leaf then the
        interface is suppressed (i.e. held down).";
    reference "RFC XXXX, Section 2.2 Dampening";
}

leaf suppressed {
    type boolean;
    config false;
    description
        "Represents whether the interface is suppressed (i.e. held
        down) because the 'penalty' leaf value exceeds the
        'suppress' leaf.";
    reference "RFC XXXX, Section 2.2 Dampening";
}

leaf time-remaining {
    when '../suppressed = "true"' {
        description
            "Only suppressed interfaces have a time remaining.";
    }
    type uint32;
    units seconds;
    config false;
    description
        "For a suppressed interface, this leaf represents how long
        (in seconds) that the interface will remain suppressed
        before it is allowed to go back up again.";
    reference "RFC XXXX, Section 2.2 Dampening";
}
}
```

```
/*
 * Various types of interfaces support a configurable layer 2
 * encapsulation, any that are supported by YANG should be
 * listed here.
 *
 * Different encapsulations can hook into the common encaps-type
 * choice statement.
 */
container encapsulation {
  when
    "derived-from-or-self(..if:type,
                          'ianaift:ethernetCsmacd') or
     derived-from-or-self(..if:type,
                          'ianaift:ieee8023adLag') or
     derived-from-or-self(..if:type, 'ianaift:pos') or
     derived-from-or-self(..if:type,
                          'ianaift:atmSubInterface') or
     derived-from-or-self(..if:type, 'ianaift:l2vlan') or
     derived-from-or-self(..if:type, 'ethSubInterface')" {

    description
      "All interface types that can have a configurable L2
       encapsulation.";
  }

  description
    "Holds the OSI layer 2 encapsulation associated with an
     interface.";
  choice encaps-type {
    description
      "Extensible choice of layer 2 encapsulations";
    reference "RFC XXXX, Section 2.3 Encapsulation";
  }
}

/*
 * Various types of interfaces support loopback configuration,
 * any that are supported by YANG should be listed here.
 */
leaf loopback {
  when "derived-from-or-self(..if:type,
                              'ianaift:ethernetCsmacd') or
       derived-from-or-self(..if:type, 'ianaift:sonet') or
       derived-from-or-self(..if:type, 'ianaift:atm') or
       derived-from-or-self(..if:type, 'ianaift:otnOtu')" {
    description
      "All interface types that support loopback configuration.";
  }
}
```

```
    if-feature "loopback";
    type identityref {
        base loopback;
    }
    description "Enables traffic loopback.";
    reference "RFC XXXX, Section 2.4 Loopback";
}

/*
 * Allows the maximum frame size to be configured or reported.
 */
leaf max-frame-size {
    if-feature "max-frame-size";
    type uint32 {
        range "64 .. max";
    }
    description
        "The maximum size of layer 2 frames that may be transmitted
        or received on the interface (including any frame header,
        maximum frame payload size, and frame checksum sequence).

        If configured, the max-frame-size also limits the maximum
        frame size of any child sub-interfaces. The MTU available
        to higher layer protocols is restricted to the maximum frame
        payload size, and MAY be further restricted by explicit
        layer 3 or protocol specific MTU configuration.";

    reference "RFC XXXX, Section 2.5 Maximum Frame Size";
}

/*
 * Augments the IETF interfaces model with a leaf that indicates
 * which mode, or layer, is being used to forward the traffic.
 */
leaf forwarding-mode {
    type identityref {
        base forwarding-mode;
    }
    config false;

    description
        "The forwarding mode that the interface is operating in.";
    reference "RFC XXXX, Section 2.7 Forwarding Mode";
}

/*
 * Add generic support for sub-interfaces.
```

```
*
* This should be extended to cover all interface types that are
* child interfaces of other interfaces.
*/
augment "/if:interfaces/if:interface" {
  when "derived-from(if:type, 'sub-interface') or
        derived-from-or-self(if:type, 'ianaift:l2vlan') or
        derived-from-or-self(if:type, 'ianaift:atmSubInterface') or
        derived-from-or-self(if:type, 'ianaift:frameRelay')" {
    description
      "Any ianaift:types that explicitly represent sub-interfaces
      or any types that derive from the sub-interface identity.";
  }
  if-feature "sub-interfaces";

  description
    "Adds a parent interface field to interfaces that model
    sub-interfaces.";
  leaf parent-interface {

    type if:interface-ref;

    mandatory true;
    description
      "This is the reference to the parent interface of this
      sub-interface.";
    reference "RFC XXXX, Section 2.6 Sub-interface";
  }
}

/*
* Add discard counter for unknown sub-interface encapsulation
*/
augment "/if:interfaces/if:interface/if:statistics" {
  when "derived-from-or-self(..if:type,
                              'ianaift:ethernetCsmacd') or
        derived-from-or-self(..if:type,
                              'ianaift:ieee8023adLag') or
        derived-from-or-self(..if:type, 'ianaift:ifPwType')" {
    description
      "Applies to interfaces that can demultiplex ingress frames to
      sub-interfaces.";
  }
  if-feature "sub-interfaces";

  description
    "Augment the interface model statistics with a sub-interface
    demux discard counter.";
```

```

leaf in-discard-unknown-encaps {
  type yang:counter64;
  units frames;
  description
    "A count of the number of frames that were well formed, but
    otherwise discarded because their encapsulation does not
    classify the frame to the interface or any child
    sub-interface. E.g., a frame might be discarded because the
    it has an unknown VLAN Id, or does not have a VLAN Id when
    one is expected.

    For consistency, frames counted against this counter are
    also counted against the IETF interfaces statistics. In
    particular, they are included in in-octets and in-discards,
    but are not included in in-unicast-pkts, in-multicast-pkts
    or in-broadcast-pkts, because they are not delivered to a
    higher layer.

    Discontinuities in the values of this counter can occur at
    re-initialization of the management system, and at other
    times as indicated by the value of the 'discontinuity-time'
    leaf defined in the ietf-interfaces YANG module
    (RFC 8343).";
}
}
}
<CODE ENDS>

```

5. Interfaces Ethernet-Like YANG Module

This YANG module augments the interface container defined in RFC 8343 [RFC8343] for Ethernet-like interfaces. This includes Ethernet interfaces, 802.3 LAG (802.1AX) interfaces, Switch Virtual interfaces, and Pseudo-Wire Head-End interfaces. It also contains references to [RFC6991], [RFC7224], and [IEEE802.3.2-2019].

```

<CODE BEGINS> file "ietf-if-ethernet-like@2019-11-04.yang"
module ietf-if-ethernet-like {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-if-ethernet-like";

  prefix ethlike;

  import ietf-interfaces {

```



```
    prefix if;
    reference
      "RFC 8343: A YANG Data Model For Interface Management";
  }

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

  import iana-if-type {
    prefix ianaift;
    reference "RFC 7224: IANA Interface Type YANG Module";
  }

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

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

    Editor:   Robert Wilton
              <mailto:rwilton@cisco.com>";

  description
    "This module contains YANG definitions for configuration for
    'Ethernet-like' interfaces.  It is applicable to all interface
    types that use Ethernet framing and expose an Ethernet MAC
    layer, and includes such interfaces as physical Ethernet
    interfaces, Ethernet LAG interfaces and VLAN sub-interfaces.

    Additional interface configuration and counters for physical
    Ethernet interfaces are defined in
    ieee802-ethernet-interface.yang, as part of IEEE Std
    802.3.2-2019.

    Copyright (c) 2019 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).

    This version of this YANG module is part of RFC XXXX
```

```
(https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
for full legal notices.";

revision 2019-11-04 {
  description "Initial revision.";

  reference
    "RFC XXXX, Common Interface Extension YANG Data Models";
}

feature configurable-mac-address {
  description
    "This feature indicates that MAC addresses on Ethernet-like
    interfaces can be configured.";
  reference
    "RFC XXXX, Section 3, Interfaces Ethernet-Like Module";
}

/*
 * Configuration parameters for Ethernet-like interfaces.
 */
augment "/if:interfaces/if:interface" {
  when "derived-from-or-self(if:type, 'ianaift:ethernetCsmacd') or
        derived-from-or-self(if:type, 'ianaift:ieee8023adLag') or
        derived-from-or-self(if:type, 'ianaift:ifPwType')" {
    description "Applies to all Ethernet-like interfaces";
  }
  description
    "Augment the interface model with parameters for all
    Ethernet-like interfaces.";

  container ethernet-like {
    description
      "Contains parameters for interfaces that use Ethernet framing
      and expose an Ethernet MAC layer.";

    leaf mac-address {
      if-feature "configurable-mac-address";
      type yang:mac-address;
      description
        "The MAC address of the interface. The operational value
        matches the /if:interfaces/if:interface/if:phys-address
        leaf defined in ietf-interface.yang.";
    }

    leaf bia-mac-address {
      type yang:mac-address;
    }
  }
}
```

```
        config false;
        description
            "The 'burnt-in' MAC address. I.e the default MAC address
            assigned to the interface if no MAC address has been
            explicitly configured on it.";
    }
}

/*
 * Configuration parameters for Ethernet-like interfaces.
 */
augment "/if:interfaces/if:interface/if:statistics" {
    when "derived-from-or-self(..if:type,
        'ianaift:ethernetCsmacd') or
        derived-from-or-self(..if:type,
        'ianaift:ieee8023adLag') or
        derived-from-or-self(..if:type, 'ianaift:ifPwType')" {
        description "Applies to all Ethernet-like interfaces";
    }
    description
        "Augment the interface model statistics with additional
        counters related to Ethernet-like interfaces.";

    leaf in-discard-unknown-dest-mac-pkts {
        type yang:counter64;
        units frames;
        description
            "A count of the number of frames that were well formed, but
            otherwise discarded because the destination MAC address did
            not pass any ingress destination MAC address filter.

            For consistency, frames counted against this counter are
            also counted against the IETF interfaces statistics. In
            particular, they are included in in-octets and in-discards,
            but are not included in in-unicast-pkts, in-multicast-pkts
            or in-broadcast-pkts, because they are not delivered to a
            higher layer.

            Discontinuities in the values of this counter can occur at
            re-initialization of the management system, and at other
            times as indicated by the value of the 'discontinuity-time'
            leaf defined in the ietf-interfaces YANG module
            (RFC 8343).";
    }
}
}
```

<CODE ENDS>

6. Examples

The following sections give some examples of how different parts of the YANG modules could be used. Examples are not given for the more trivial configuration, or for sub-interfaces, for which examples are contained in [I-D.ietf-netmod-sub-intf-vlan-model].

6.1. Carrier delay configuration

The following example shows how the operational state datastore could look like for an Ethernet interface without any carrier delay configuration. The down leaf value of 0 indicates that link down events as always propagated to high layers immediately, but an up leaf value of 50 indicates that the interface must be up and stable for at least 50 msec before the interface is reported as being up to the high layers.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
  xmlns:if-ext="urn:ietf:params:xml:ns:yang:ietf-if-extensions">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <if-ext:carrier-delay>
      <if-ext:down>0</if-ext:down>
      <if-ext:up>50</if-ext:up>
    </if-ext:carrier-delay>
  </interface>
</interfaces>
```

The following example shows explicit carrier delay up and down values have been configured. A 50 msec down leaf value has been used to potentially allow optical protection to recover the link before the higher layer protocol state is flapped. A 1 second (1000 milliseconds) up leaf value has been used to ensure that the link is always reasonably stable before allowing traffic to be carried over it. This also has the benefit of greatly reducing the rate at which higher layer protocol state flaps could occur.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
    xmlns:if-ext="urn:ietf:params:xml:ns:yang:ietf-if-extensions">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
      <if-ext:carrier-delay>
        <if-ext:down>50</if-ext:down>
        <if-ext:up>1000</if-ext:up>
      </if-ext:carrier-delay>
    </interface>
  </interfaces>
</config>
```

6.2. Dampening configuration

The following example shows what the operational state datastore may look like for an interface configured with interface dampening. The 'suppressed' leaf indicates that the interface is currently suppressed (i.e. down) because the 'penalty' is greater than the 'suppress' leaf threshold. The 'time-remaining' leaf indicates that the interface will remain suppressed for another 103 seconds before the 'penalty' is below the 'reuse' leaf value and the interface is allowed to go back up again.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <oper-status>down</oper-status>
    <dampening
      xmlns="urn:ietf:params:xml:ns:yang:ietf-if-extensions">
        <half-life>60</half-life>
        <reuse>750</reuse>
        <suppress>2000</suppress>
        <max-suppress-time>240</max-suppress-time>
        <penalty>2480</penalty>
        <suppressed>true</suppressed>
        <time-remaining>103</time-remaining>
      </dampening>
    </interface>
  </interfaces>
```

6.3. MAC address configuration

The following example shows how the operational state datastore could look like for an Ethernet interface without an explicit MAC address configured. The mac-address leaf always reports the actual operational MAC address that is in use. The bia-mac-address leaf always reports the default MAC address assigned to the hardware.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <phys-address>00:00:5E:00:53:30</phys-address>
    <ethernet-like
      xmlns="urn:ietf:params:xml:ns:yang:ietf-if-ethernet-like">
        <mac-address>00:00:5E:00:53:30</mac-address>
        <bia-mac-address>00:00:5E:00:53:30</bia-mac-address>
      </ethernet-like>
    </interface>
  </interfaces>
```

The following example shows the intended configuration for interface eth0 with an explicit MAC address configured.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
      <ethernet-like
        xmlns="urn:ietf:params:xml:ns:yang:ietf-if-ethernet-like">
        <mac-address>00:00:5E:00:53:35</mac-address>
      </ethernet-like>
    </interface>
  </interfaces>
</config>
```

After the MAC address configuration has been successfully applied, the operational state datastore reporting the interface MAC address properties would contain the following, with the mac-address leaf updated to match the configured value, but the bia-mac-address leaf retaining the same value - which should never change.

```
<?xml version="1.0" encoding="utf-8"?>
<interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
    <phys-address>00:00:5E:00:53:35</phys-address>
    <ethernet-like
      xmlns="urn:ietf:params:xml:ns:yang:ietf-if-ethernet-like">
      <mac-address>00:00:5E:00:53:35</mac-address>
      <bia-mac-address>00:00:5E:00:53:30</bia-mac-address>
    </ethernet-like>
  </interface>
</interfaces>
```

7. Acknowledgements

The authors wish to thank Eric Gray, Ing-Wher Chen, Jon Culver, Juergen Schoenwaelder, Ladislav Lhotka, Lou Berger, Mahesh Jethanandani, Martin Bjorklund, Michael Zitao, Neil Ketley, Qin Wu, William Lupton, Xufeng Liu, Andy Bierman, and Vladimir Vassilev for their helpful comments contributing to this document.

8. ChangeLog

XXX, RFC Editor, please delete this change log before publication.

8.1. Version -10

- o Update modules from github and tree diagram.

8.2. Version -09

- o Fixed IANA section.

8.3. Version -08

- o Initial updates after WG LC comments.

8.4. Version -07

- o Minor editorial updates

8.5. Version -06

- o Remove reservable-bandwidth, based on Acee's suggestion
- o Add examples
- o Add additional state parameters for carrier-delay and dampening

8.6. Version -05

- o Incorporate feedback from Andy Bierman

8.7. Version -04

- o Incorporate feedback from Lada, some comments left as open issues.

8.8. Version -03

- o Fixed incorrect module name references, and updated tree output

8.9. Version -02

- o Minor changes only: Fix errors in when statements, use derived-from-or-self() for future proofing.

9. IANA Considerations

9.1. YANG Module Registrations

The following YANG modules are requested to be registered in the IANA "YANG Module Names" [RFC6020] registry:

The ietf-if-extensions module:

Name: ietf-if-extensions

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-if-extensions

Prefix: if-ext

Reference: [RFCXXXX]

The ietf-if-ethernet-like module:

Name: ietf-if-ethernet-like

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-if-ethernet-like

Prefix: ethlike

Reference: [RFCXXXX]

This document registers two URIs in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registrations have been made.

URI: urn:ietf:params:xml:ns:yang:ietf-if-extensions

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-if-ethernet-like

Registrant Contact: The IESG.

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

10. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [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 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:

10.1. ietf-if-extensions.yang

The ietf-if-extensions YANG module contains various configuration leaves that affect the behavior of interfaces. Modifying these leaves can cause an interface to go down, or become unreliable, or to drop traffic forwarded over it. More specific details of the possible failure modes are given below.

The following leaf could cause the interface to go down and stop processing any ingress or egress traffic on the interface. It could also cause broadcast traffic storms.

- o /if:interfaces/if:interface/loopback

The following leaves could cause instabilities at the interface link layer, and cause unwanted higher layer routing path changes if the leaves are modified, although they would generally only affect a device that had some underlying link stability issues:

- o /if:interfaces/if:interface/carrier-delay/down
- o /if:interfaces/if:interface/carrier-delay/up
- o /if:interfaces/if:interface/dampening/half-life
- o /if:interfaces/if:interface/dampening/reuse

- o /if:interfaces/if:interface/dampening/suppress
- o /if:interfaces/if:interface/dampening/max-suppress-time

The following leaves could cause traffic loss on the interface because the received or transmitted frames do not comply with the frame matching criteria on the interface and hence would be dropped:

- o /if:interfaces/if:interface/encapsulation
- o /if:interfaces/if:interface/max-frame-size
- o /if:interfaces/if:interface/forwarding-mode

Changing the parent-interface leaf could cause all traffic on the affected interface to be dropped. The affected leaf is:

- o /if:interfaces/if:interface/parent-interface

10.2. ietf-if-ethernet-like.yang

Generally, the configuration nodes in the ietf-if-ethernet-like YANG module are concerned with configuration that is common across all types of Ethernet-like interfaces. The module currently only contains a node for configuring the operational MAC address to use on an interface. Adding/modifying/deleting this leaf has the potential risk of causing protocol instability, excessive protocol traffic, and general traffic loss, particularly if the configuration change caused a duplicate MAC address to be present on the local network. The following leaf is affected:

- o interfaces/interface/ethernet-like/mac-address

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>>.
- [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>>.
- [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>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.

11.2. Informative References

- [I-D.ietf-netmod-sub-intf-vlan-model]
Wilton, R., Ball, D., tapsingh@cisco.com, t., and S. Sivaram, "Sub-interface VLAN YANG Data Models", draft-ietf-netmod-sub-intf-vlan-model-07 (work in progress), July 2020.
- [IEEE802.3.2-2019]
IEEE WG802.3 - Ethernet Working Group, "IEEE 802.3.2-2019", 2019.
- [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>>.

- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<https://www.rfc-editor.org/info/rfc7224>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@cisco.com

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

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

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
July 3, 2017

Network Management Datastore Architecture
draft-ietf-netmod-revised-datastores-03

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

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Background	5
3.1. Original Model of Datastores	6
4. Architectural Model of Datastores	8
4.1. The Startup Configuration Datastore (<startup>)	9
4.2. The Candidate Configuration Datastore (<candidate>)	10
4.3. The Running Configuration Datastore (<running>)	10
4.4. The Intended Configuration Datastore (<intended>)	10
4.5. Conventional Configuration Datastores	11
4.6. Dynamic Datastores	11
4.7. The Operational State Datastore (<operational>)	11
4.7.1. Missing Resources	12
4.7.2. System-controlled Resources	13
4.7.3. Origin Metadata Annotation	13
5. Implications on YANG	14
5.1. XPath Context	14
6. YANG Modules	15
7. IANA Considerations	21
7.1. Updates to the IETF XML Registry	21
7.2. Updates to the YANG Module Names Registry	22
8. Security Considerations	22
9. Acknowledgments	22
10. References	23
10.1. Normative References	23
10.2. Informative References	23
Appendix A. Guidelines for Defining Datastores	24
A.1. Define which YANG modules can be used in the datastore	24
A.2. Define which subset of YANG-modeled data applies	25
A.3. Define how data is actualized	25
A.4. Define which protocols can be used	25
A.5. Define YANG identities for the datastore	25
Appendix B. Ephemeral Dynamic Datastore Example	25
Appendix C. Example Data	27
C.1. System Example	27
C.2. BGP Example	29
C.2.1. Datastores	31
C.2.2. Adding a Peer	31

C.2.3. Removing a Peer	32
C.3. Interface Example	33
C.3.1. Pre-provisioned Interfaces	33
C.3.2. System-provided Interface	34
Authors' Addresses	35

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.

2. Terminology

This document defines the following terms:

- 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 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.
- o configuration datastore: A datastore holding configuration.
- o running configuration datastore: A configuration datastore holding the current configuration of the device. It may include inactive configuration or template-mechanism-oriented configuration that require further expansion. This datastore is commonly referred to as "<running>".
- o 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 commonly referred to as "<candidate>".

- o startup configuration datastore: A configuration datastore holding the configuration loaded by the device into the running configuration datastore when it boots. This datastore is commonly referred to as "<startup>".
- o intended configuration: Configuration that is intended to be used by the device. For example, intended configuration excludes any inactive configuration and it would include configuration produced through the expansion of templates.
- o intended configuration datastore: A configuration datastore holding the complete intended configuration of the device. This datastore is commonly referred to as "<intended>".
- o conventional configuration datastore: One of the following set of configuration datastores: <running>, <startup>, <candidate>, and <intended>. These datastores share a common schema and protocol operations allow copying data between these datastores. The term "conventional" is chosen as a generic umbrella term for these datastores.
- o conventional configuration: Configuration that is stored in any of the conventional configuration datastores.
- o dynamic datastore: A datastore holding data obtained dynamically during the operation of a device through interaction with other systems, rather than through one of the conventional configuration datastores.
- o dynamic configuration: Configuration obtained via a dynamic datastore.
- o learned configuration: Configuration that has been learned via protocol interactions with other systems that is not conventional or dynamic configuration.
- o system configuration: Configuration that is supplied by the device itself.
- o default configuration: Configuration that is not explicitly provided but for which a value defined in the data model is used.
- o applied configuration: Configuration that is actively in use by a device. Applied configuration originates from conventional, dynamic, learned, system and default configuration.
- o 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.

- o operational state: The combination of applied configuration and system state.
- o operational state datastore: A datastore holding the complete operational state of the device. This datastore is commonly referred to as "<operational>".
- o origin: A metadata annotation indicating the origin of a data item.
- o 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.

3. 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.

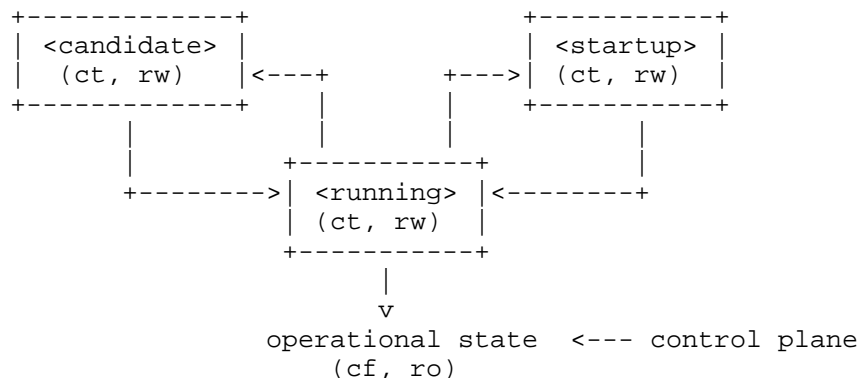
Section 4.3.3 of [RFC6244] discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of this document then concludes that at the time of the writing, modeling state as distinct leafs and distinct branches is 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.

Furthermore, separating operational state from configuration in a separate branch in the data model has been found operationally complicated, and typically impacts the readability of module definitions due to overuse of groupings. The relationship between the branches is not machine readable and filter expressions operating on configuration and on related operational state are different.

3.1. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:



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:

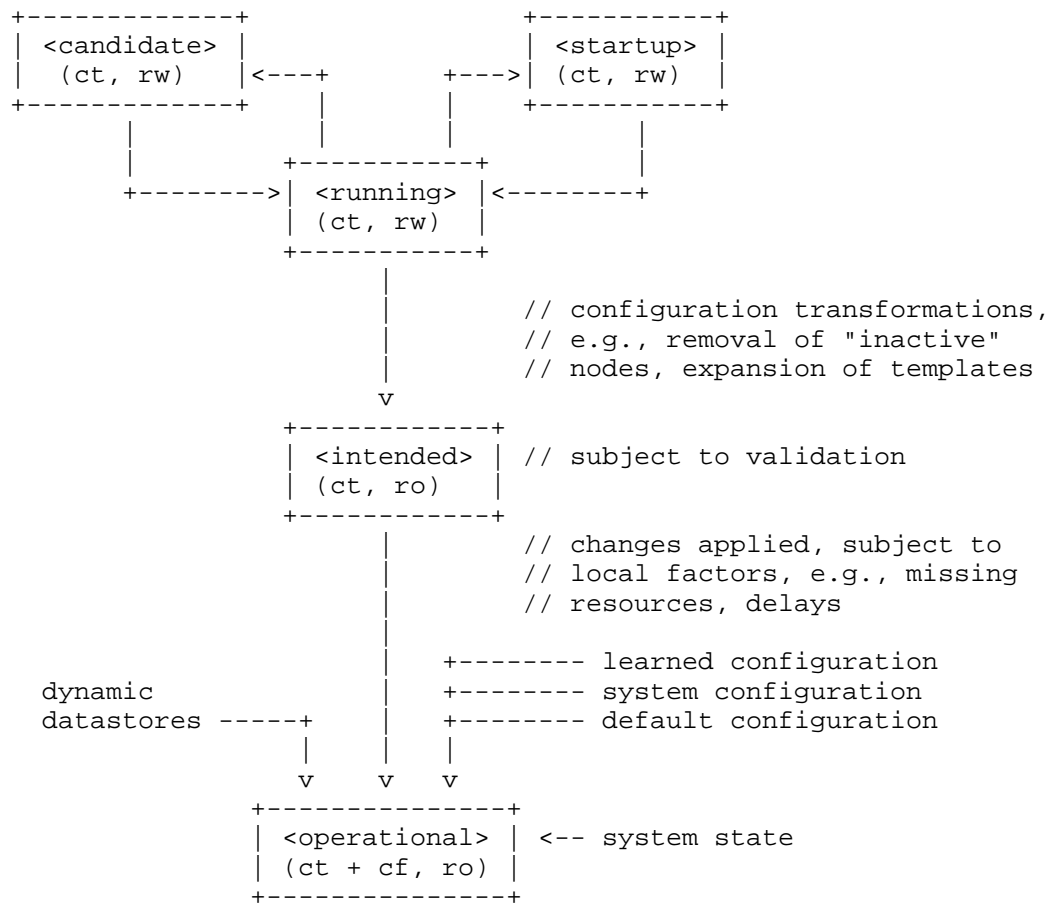
- o Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.
- o The NETCONF <get/> operation returns the content of the <running> configuration datastore together with the operational state. It is therefore necessary 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>; this inactive data is only exposed to clients that indicate that they support the

concept of inactive data; clients not indicating support for inactive data receive the content of <running> with the inactive data removed. 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.

4. 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.



```
ct = config true; cf = config false
rw = read-write; ro = read-only
boxes denote named datastores
```

4.1. The Startup Configuration Datastore (<startup>)

The startup configuration datastore (<startup>) is an optional 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.

4.2. The Candidate Configuration Datastore (`<candidate>`)

The candidate configuration datastore (`<candidate>`) is an optional 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 should be reset at boot time to the contents of `<running>`.

4.3. The Running Configuration Datastore (`<running>`)

The running configuration datastore (`<running>`) holds the complete current configuration on the device. It may include inactive configuration or template-mechanism-oriented configuration that require further expansion.

If a device does not have a distinct `<startup>` and non-volatile is available, the device will typically use that non-volatile storage to allow `<running>` to persist across reboots.

4.4. The Intended Configuration Datastore (`<intended>`)

The intended configuration datastore (`<intended>`) is a read-only configuration datastore. It is tightly coupled to `<running>`. When data is written to `<running>`, the data that is to be validated is also conceptually written to `<intended>`. Validation is performed on the contents of `<intended>`.

For simple implementations, `<running>` and `<intended>` are identical.

`<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 contents 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>, and are thus not taken into account when validating the configuration.

Another example is support for templates. Templates are expanded when copied into <intended>, and the expanded result is validated.

4.5. Conventional Configuration Datastores

The conventional configuration datastores are a set of configuration datastores that share a common 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 Section 4.

The specific protocols may define explicit operations to copy between these datastores, e.g., NETCONF's <copy-config> operation.

4.6. Dynamic Datastores

The model recognizes the need for dynamic 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 datastores interact with the rest of the system through <operational>.

4.7. 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 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>, system-provided configuration, and default values defined by any supported data models. In addition, <operational> also contains applied data from dynamic datastores.

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.

<operational> does not persist across reboots.

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).

As a result of remnant configuration, 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. The constraints that may not hold include "when", "must", "min-elements", and "max-elements". Note that syntactic constraints cannot be violated, including hierarchical organization, identifiers, and type-based constraints.

4.7.1. Missing Resources

Configuration in <intended> can refer to resources that are not available or otherwise not physically present. In these situations, these parts of the <intended> configuration 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 the removing a resource might render the configuration invalid. This is unacceptable, especially given that rebooting such a device would fail to boot due to an invalid configuration. Instead we allow configuration for missing resources to exist in <running> and <intended>, but it will not appear in <operational>.

4.7.2. System-controlled Resources

Sometimes resources are controlled by the device and the corresponding system controlled data appear in (and disappear 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).

4.7.3. Origin Metadata Annotation

As data flows into <operational>, it is conceptually marked with a metadata annotation ([RFC7952]) that indicates its origin. The origin applies to all data nodes except non-presence containers. The "origin" metadata annotation is defined in Section 6. 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 data provided by <intended>.
- o dynamic: represents data provided by a dynamic datastore.
- o system: represents data provided by the system itself, including both system configuration and system state. 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 data 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 data has not been provided by any other source.

- o unknown: represents data 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 datastore derived from "dynamic".

In all cases, the device should report the origin that most accurately reflects the source of the data that is actively being used 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 data node then it should use the origin "unknown".

5. Implications on YANG

5.1. XPath Context

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. YANG Modules

<CODE BEGINS> file "ietf-datastores@2017-04-26.yang"

```
module ietf-datastores {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-datastores";
  prefix ds;

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) 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 are for datastore propterties.

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 2017-04-26 {  
  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 datastores.";
}

identity operational {
  base datastore;
  description
    "The operational state datastore.";
}

identity property {
  description
    "Abstract base identity for datastore identities.";
}

identity writable {
  base property;
  description
    "Used on the 'running' datastore to indicate that it can be
    written to.";
```

```
}  
  
identity auto-persist {  
  base property;  
  description  
    "Used on the 'running' datastore to indicate that writes to  
    it will be automatically persisted.  
  
    If the 'startup' datastore is also supported, clients may  
    query its contents to ensure its synchronization.  
  
    If the 'startup' datastore is not supported, and this  
    property is not set, then clients must use a mechanism  
    provided by the protocol to explicitly persist the  
    'running' datastore's contents.";  
}  
  
identity rollback-on-error {  
  base property;  
  description  
    "Used on either the 'running' or 'candidate' datastores to  
    indicate that clients may request atomic update behavior.";  
}  
  
identity confirmed-commit {  
  base property;  
  description  
    "Used on the 'candidate' datastore to indicate that  
    clients may request confirmed-commit update behavior.";  
}  
  
identity validate {  
  base property;  
  description  
    "Used on the 'candidate' datastore to indicate that  
    clients may request datastore validation.";  
}  
}
```

<CODE ENDS>

<CODE BEGINS> file "ietf-origin@2017-04-26.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 NETMOD (NETCONF Data Modeling Language) 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 2017-04-26 {
```



```
    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 data from the intended configuration datastore";
}

identity dynamic {
    base origin;
    description
        "Denotes data from a dynamic datastore.";
}

identity system {
    base origin;
    description
        "Denotes data originated by the system itself, including
        both system configuration and system state.

        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 the intended configuration
        datastore or any dynamic datastore.

        Examples of protocols that provide learned configuration
        include link-layer negotiations, routing protocols, and
```

```

        DHCP.";
    }

    identity default {
        base origin;
        description
            "Denotes data 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 data for which the system cannot identify the
            origin.";
    }

    /*
     * Metadata annotations
     */

    md:annotation origin {
        type identityref {
            base origin;
        }
        description
            "The 'origin' annotation can be present on any node in a
            datastore.  It specifies from where the node originated.";
    }
}

<CODE ENDS>

```

7. IANA Considerations

7.1. 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.

7.2. 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

8. 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.

9. 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 Acee Lindem, Cisco Systems, <acee@cisco.com>
- o Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- o Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- o Anees Shaikh, Google, <aashaikh@google.com>
- o Rob Shakir, Google, <robjs@google.com>

Juergen Schoenwaelder was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

10. References

10.1. Normative References

- [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>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <<http://www.rfc-editor.org/info/rfc7952>>.
- [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>>.

10.2. 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, <<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>>.
- [RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <<http://www.rfc-editor.org/info/rfc6244>>.

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 are present, or "config false nodes" that also have a specific YANG extension (e.g., "i2rs:ephemeral true") 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 4 depicts dynamic datastores feeding into <operational>. How this interaction occurs must be defined by any dynamic datastore. In some cases, it may occur implicitly, as soon as the data is put into the dynamic 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 6.

An example of these guidelines in use is provided in Appendix B.

Appendix B. Ephemeral Dynamic Datastore Example

The section defines documentation for an example dynamic 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 datastore called "ephemeral", which is loosely modeled after the work done in the I2RS working group.

1. Name : ephemeral
2. YANG modules : all (default)
3. YANG statements : config false + ephemeral true
4. How applied : automatic
5. Protocols : NC/RC (default)
6. YANG Module : (see below)

```
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;
  }

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

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

  // define ephemeral extension
  extension ephemeral {
    argument "value";
    description
      "This extension is mixed into config false YANG nodes to
      indicate that they are writable nodes in the 'ephemeral'
      datastore. This statement takes a single argument
      representing a boolean having the values 'true' and
      'false'. The default value is 'false'.";
  }
}
```

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> is:

```

<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>32</prefix-length>
        </address>
    </interface>

    <interface>
        <name>eth1</name>
        <address>
            <ip>2001:db8::20</ip>
            <prefix-length>32</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>:

```
<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 piece of a ersatz 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 not 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 user, 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" feature. 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>64642</local-as>
  <peer-as>65000</peer-as>
  <peer>
    <name>10.1.2.3</name>
  </peer>
</bgp>
```

C.2.2.1. <operational>

<operational> 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 origin="or:intended" xmlns="urn:example:bgp">
  <local-as origin="or:intended">64642</local-as>
  <peer-as origin="or:intended">65000</peer-as>
  <peer origin="or:intended">
    <name origin="or:intended">10.1.2.3</name>
    <local-as origin="or:default">64642</local-as>
    <peer-as origin="or:default">65000</peer-as>
    <local-port origin="or:system">60794</local-port>
    <remote-port origin="or:default">179</remote-port>
  </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 origin="or:intended">
  <local-as origin="or:intended">64642</local-as>
  <peer-as origin="or:intended">65000</peer-as>
  <peer origin="or:intended">
    <name origin="or:intended">10.1.2.3</name>
    <local-as origin="or:default">64642</local-as>
    <peer-as origin="or:default">65000</peer-as>
    <local-port origin="or:system">60794</local-port>
    <remote-port origin="or:default">179</remote-port>
  </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'll use this simple interface data model:

```

container interfaces {
  list interface {
    key name;
    leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf mtu {
      type uint;
    }
    leaf ipv4-address {
      type inet:ipv4-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. The <operational> will contain the data from <intended>, as well as the "config false" nodes, such as the current value of the interface's MTU.

```

<interfaces origin="or:intended">
  <interface origin="or:intended">
    <name origin="or:intended">et-0/0/0</name>
    <description origin="or:intended">Test interface</description>
    <mtu 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 ipv4-address of "127.0.0.1". 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 origin="or:intended">
  <interface origin="or:system">
    <name origin="or:system">lo0</name>
    <ipv4-address origin="or:system">127.0.0.1</ipv4-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 "ipv4-address" is not provided, then the system-provided value will appear as follows:

```
<interfaces origin="or:intended">
  <interface origin="or:intended">
    <name origin="or:intended">lo0</name>
    <description origin="or:intended">loopback</description>
    <ipv4-address origin="or:system">127.0.0.1</ipv4-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

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

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

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

This Internet-Draft will expire on July 17, 2018.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	3
2. Objectives	3
3. Terminology	4
4. Background	7
4.1. Original Model of Datastores	8
5. Architectural Model of Datastores	9
5.1. Conventional Configuration Datastores	10
5.1.1. The Startup Configuration Datastore (<startup>) . . .	11
5.1.2. The Candidate Configuration Datastore (<candidate>) .	11
5.1.3. The Running Configuration Datastore (<running>) . . .	12
5.1.4. The Intended Configuration Datastore (<intended>) . .	12
5.2. Dynamic Configuration Datastores	13
5.3. The Operational State Datastore (<operational>)	13
5.3.1. Remnant Configuration	14
5.3.2. Missing Resources	15
5.3.3. System-controlled Resources	15
5.3.4. Origin Metadata Annotation	15
6. Implications on YANG	17
6.1. XPath Context	17
6.2. Invocation of Actions and RPCs	18
7. YANG Modules	18
8. IANA Considerations	24
8.1. Updates to the IETF XML Registry	24
8.2. Updates to the YANG Module Names Registry	24
9. Security Considerations	24
10. Acknowledgments	25
11. References	25
11.1. Normative References	26
11.2. Informative References	26
Appendix A. Guidelines for Defining Datastores	27
A.1. Define which YANG modules can be used in the datastore .	27
A.2. Define which subset of YANG-modeled data applies	28
A.3. Define how data is actualized	28
A.4. Define which protocols can be used	28
A.5. Define YANG identities for the datastore	28
Appendix B. Ephemeral Dynamic Configuration Datastore Example .	29
Appendix C. Example Data	30
C.1. System Example	30

C.2. BGP Example	33
C.2.1. Datastores	35
C.2.2. Adding a Peer	35
C.2.3. Removing a Peer	36
C.3. Interface Example	37
C.3.1. Pre-provisioned Interfaces	37
C.3.2. System-provided Interface	38
Authors' Addresses	39

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.

- 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.
- o configuration datastore: A datastore holding configuration.
- o 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>".
- o 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>".
- o 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>".
- o 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.
- o intended configuration datastore: A configuration datastore holding the complete intended configuration of the device. This datastore is referred to as "<intended>".
- o 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.
- o 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.

- o conventional configuration: Configuration that is stored in any of the conventional configuration datastores.
- o 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.
- o dynamic configuration: Configuration obtained via a dynamic configuration datastore.
- o learned configuration: Configuration that has been learned via protocol interactions with other systems and that is neither conventional nor dynamic configuration.
- o system configuration: Configuration that is supplied by the device itself.
- o default configuration: Configuration that is not explicitly provided but for which a value defined in the data model is used.
- o applied configuration: Configuration that is actively in use by a device. Applied configuration originates from conventional, dynamic, learned, system and default configuration.
- o 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.
- o operational state: The combination of applied configuration and system state.
- o operational state datastore: A datastore holding the complete operational state of the device. This datastore is referred to as "<operational>".
- o origin: A metadata annotation indicating the origin of a data item.
- o 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.

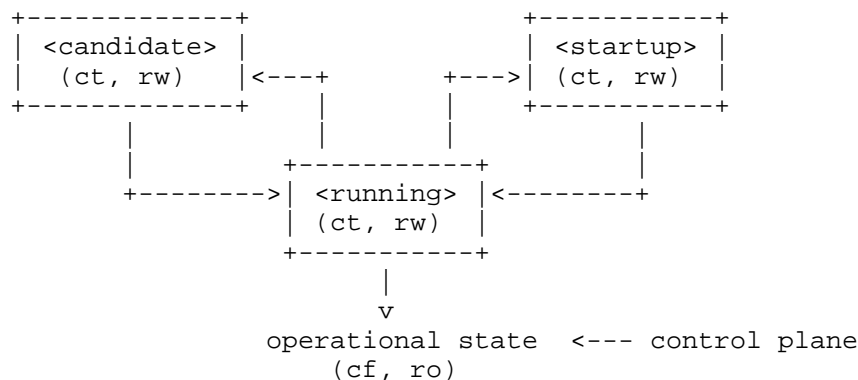
Section 4.3.3 of [RFC6244] discusses operational state and among other things mentions the option to consider operational state as

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]:



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.

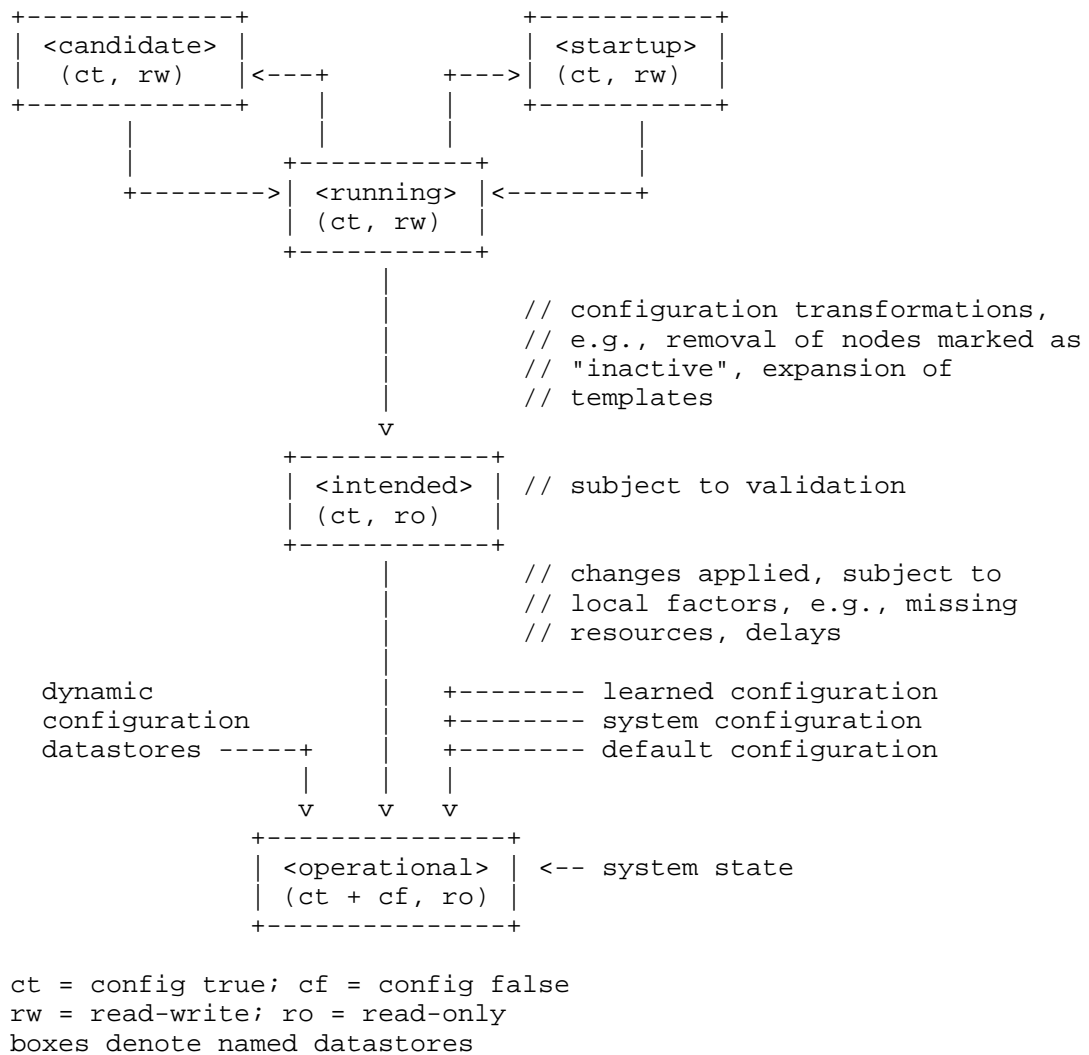


Figure 2

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:

- 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, <ietf@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.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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2018

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC
July 5, 2017

YANG Schema Mount
draft-ietf-netmod-schema-mount-06

Abstract

This document defines a mechanism to combine YANG modules into the schema defined in other YANG modules.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 6, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Terminology and Notation	5
2.1. Glossary of New Terms	6
2.2. Namespace Prefixes	6
3. Schema Mount	6
3.1. Mount Point Definition	7
3.2. Specification of the Mounted Schema	7
3.3. Multiple Levels of Schema Mount	9
4. Referring to Data Nodes in the Parent Schema	9
5. RPC operations and Notifications	10
6. Implementation Notes	11
7. Data Model	11
8. Schema Mount YANG Module	13
9. IANA Considerations	18
10. Security Considerations	18
11. Contributors	18
12. References	19
12.1. Normative References	19
12.2. Informative References	19
Appendix A. Example: Device Model with LNEs and NIs	20
A.1. Physical Device	21
A.2. Logical Network Elements	22
A.3. Network Instances	25
A.4. Invoking an RPC Operation	26
Authors' Addresses	27

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:

- o client
- o notification
- o server

The following terms are defined in [RFC7950] and are not redefined here:

- o action
- o configuration data
- o container
- o list
- o operation

The following terms are defined in [RFC7223] and are not redefined here:

- o 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 the name of 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 the name of 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.

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 that 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 root of the parent data tree as the context node and the result 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 name]
      | +--ro module      yang:yang-identifier
      | +--ro name        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 name]
        +--ro module      yang:yang-identifier
        +--ro name        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@2017-06-16.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 2017-06-16 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: YANG Schema Mount";
}
```

```
/*
 * Extensions
 */
```

```
extension mount-point {
  argument name;
  description
    "The argument 'name' 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.

If a mount point is defined in a grouping, its name 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.";
}

/*
 * 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 name";
    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 name {
      type yang:yang-identifier;
      description
        "Name 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 {
  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";
    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 root node of the parent data
         tree.

       - 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

TBD

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:

- 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>

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",
      "name": "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"
    }
  ],
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "name": "root",
      "use-schema": [
        {
          "name": "ni-schema",
          "parent-reference": ["/if:interfaces"]
        }
      ]
    }
  ]
}
```



```

    ]
  }
],
"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

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

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

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

This Internet-Draft will expire on April 20, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Terminology and Notation	5
2.1. Tree Diagrams	6
2.2. Namespace Prefixes	6
3. Schema Mount	7
3.1. Mount Point Definition	7
3.2. Data Model	8
3.3. Specification of the Mounted Schema	8
3.4. Multiple Levels of Schema Mount	9
4. Referring to Data Nodes in the Parent Schema	9
5. RPC operations and Notifications	11
6. Network Management Datastore Architecture (NMDA) Considerations	11
7. Interaction with the Network Configuration Access Control Model (NACM)	11
8. Implementation Notes	12
9. Schema Mount YANG Module	12
10. IANA Considerations	17
11. Security Considerations	17
12. Contributors	18
13. References	19
13.1. Normative References	19
13.2. Informative References	20
Appendix A. Example: Device Model with LNEs and NIs	21
A.1. Physical Device	21
A.2. Logical Network Elements	23
A.3. Network Instances	26
A.4. Invoking an RPC Operation	27
Authors' Addresses	28

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:

- o action
- o container
- o data node
- o list
- o RPC operation
- o schema node
- o 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](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](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](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](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 '\n' 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[\n
              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

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 4, 2018

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
July 3, 2017

Sub-interface VLAN YANG Data Models
draft-ietf-netmod-sub-intf-vlan-model-02

Abstract

This document defines YANG modules to add support for classifying traffic received on interfaces as Ethernet/VLAN framed packets to sub-interfaces based on the fields available in the Ethernet/VLAN frame headers. These modules allow configuration of Layer 3 and Layer 2 sub-interfaces (e.g. attachment circuits) that can interoperate with IETF based forwarding protocols; such as IP and L3VPN services; or L2VPN services like VPWS, VPLS, and EVPN. The sub-interfaces also interoperate with VLAN tagged traffic originating from an IEEE 802.1Q compliant bridge. Primarily the classification is based on VLAN identifiers in the 802.1Q VLAN tags, but the model also has support for matching on some other layer 2 frame header fields and is designed to be extensible to match on other arbitrary header fields.

The model differs from an IEEE 802.1Q bridge model in that the configuration is interface/sub-interface based as opposed to being based on membership of an 802.1Q VLAN bridge.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Tree Diagrams	4
2. Objectives	4
2.1. Interoperability with IEEE 802.1Q compliant bridges	4
2.2. Extensibility	4
3. L3 Interface VLAN Model	5
4. Flexible Encapsulation Model	5
5. L3 Interface VLAN YANG Module	8
6. Flexible Encapsulation YANG Module	11
7. Open Issues	19
8. Acknowledgements	20
9. ChangeLog	20
9.1. WG version -02	20
9.2. WG version -01	20
9.3. Version -04	20
9.4. Version -03	20
10. IANA Considerations	21
11. Security Considerations	21
11.1. if-l3-vlan.yang	21
11.2. flexible-encapsulation.yang	22
12. References	24
12.1. Normative References	24
12.2. Informative References	24
Appendix A. Comparison with the IEEE 802.1Q Configuration Model	25
A.1. Sub-interface based configuration model overview	25
A.2. IEEE 802.1Q Bridge Configuration Model Overview	26
A.3. Possible Overlap Between the Two Models	26
Authors' Addresses	27

1. Introduction

This document defines two YANG [RFC7950] modules that augment the encapsulation choice YANG element defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang] and the generic interfaces data model defined in [RFC7223]. The two modules provide configuration nodes to support classification of Ethernet/VLAN traffic to sub-interfaces, that can have interface based feature and service configuration applied to them.

The purpose of these models is to allow IETF defined forwarding protocols, such as IPv6 [RFC2460], Ethernet Pseudo Wires [RFC4448] and VPLS [RFC4761] [RFC4762] to be configurable via YANG when interoperating with VLAN tagged traffic received from an IEEE 802.1Q compliant bridge.

In the case of layer 2 Ethernet services, the flexible encapsulation module also supports flexible rewriting of the VLAN tags contained the in frame header.

For reference, a comparison between the sub-interface based YANG model documented in this draft and an IEEE 802.1Q bridge model is described in Appendix A.

In summary, the YANG modules defined in this internet draft are:

if-l3-vlan.yang - Defines the model for basic classification of VLAN tagged traffic to L3 transport services

flexible-encapsulation.yang - Defines the model for flexible classification of Ethernet/VLAN traffic to L2 transport services

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 RFC 2119 [RFC2119].

Sub-interface: A sub-interface is a small augmentation of a regular interface in the standard YANG module for Interface Management that represents a subset of the traffic handled by its parent interface. As such, it supports both configuration and operational data using any other YANG models that augment or reference interfaces in the normal way. It is defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration (read-write), and "ro" means state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, 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. Objectives

The primary aim of the YANG modules contained in this draft is to provide the core model that is required to implement VLAN transport services on router based devices that is fully compatible with IEEE 802.1Q compliant bridges.

A secondary aim is for the modules to be structured in such a way that they can be cleanly extended in future.

2.1. Interoperability with IEEE 802.1Q compliant bridges

The modules defined in this document are designed to fully interoperate with IEEE 802.1Q compliant bridges. In particular, the models are restricted to only matching, adding, or rewriting the 802.1Q VLAN tags in frames in ways that are compatible with IEEE 802.1Q compliant bridges.

2.2. Extensibility

The modules are structured in such a way that they can be sensibly extended. In particular:

The tag stack is represented as a list to allow a tag stack of more than two tags to be supported if necessary in future.

The tag stack list elements allow other models, or vendors, to include additional forms of tag matching and rewriting. The

intention, however, is that it should not be necessary to have any vendor specific extensions to any of the YANG models defined in this document to implement standard Ethernet and VLAN services.

3. L3 Interface VLAN Model

The L3 Interface VLAN model provides appropriate leaves for termination of an 802.1Q VLAN tagged segment to a sub-interface based L3 service. It allows for termination of traffic with up to two 802.1Q VLAN tags.

The "if-l3-vlan" YANG module has the following structure:

```
module: ietf-if-l3-vlan
  augment /if:interfaces/if:interface/if-cmn:encapsulation/
                                              if-cmn:encaps-type:
    +---:(dot1q-vlan)
      +--rw dot1q-vlan
        +--rw outer-tag!
          |   +--rw dot1q-tag
          |   |   +--rw tag-type      dot1q-tag-type
          |   |   +--rw vlan-id      ieee:vlanid
          +--rw second-tag!
            +--rw dot1q-tag
              +--rw tag-type      dot1q-tag-type
              +--rw vlan-id      ieee:vlanid
```

4. Flexible Encapsulation Model

The Flexible Encapsulation model is designed to allow for the flexible provisioning of layer 2 services. It provides the capability to classify Ethernet/VLAN frames received on an Ethernet trunk interface to sub-interfaces based on the fields available in the layer 2 headers. Once classified to sub-interfaces, it provides the capability to selectively modify fields within the layer 2 headers before the frame is handed off to the appropriate forwarding code for further handling.

The model supports a common core set of layer 2 header matches based on the 802.1Q tag type and VLAN Ids contained within the header up to a tag stack depth of two tags.

The model supports flexible rewrites of the layer 2 frame header for data frames as they are processed on the interface. It defines a set of standard tag manipulations that allow for the insertion, removal, or rewrite of one or two 802.1Q VLAN tags. The expectation is that

manipulations are generally implemented in a symmetrical fashion, i.e. if a manipulation is performed on ingress traffic on an interface then the reverse manipulation is always performed on egress traffic out of the same interface. However, the model also allows for asymmetrical rewrites, which may be required to implement some forwarding models (such as E-Tree).

The structure of the model is currently limited to matching or rewriting a maximum of two 802.1Q tags in the frame header but has been designed to be easily extensible to matching/rewriting three or more VLAN tags in future, if required.

The final aim for the model design is for it to be cleanly extensible to add in additional match and rewrite criteria of the layer 2 header, such as matching on the source or destination MAC address, PCP or DEI fields in the 802.1Q tags, or the EtherType of the frame payload. Rewrites can also be extended to allow for modification of other fields within the layer 2 frame header.

The "flexible-encapsulation" YANG module has the following structure:

```

module: ietf-flexible-encapsulation
  augment /if:interfaces/if:interface/if-cmn:encapsulation/
    if-cmn:encaps-type:
      +--:(flexible)
        +--rw flexible
          +--rw match
            +--rw (match-type)
              +--:(default)
                | +--rw default?          empty
              +--:(untagged)
                | +--rw untagged?          empty
              +--:(dot1q-priority-tagged)
                | +--rw dot1q-priority-tagged
                |   +--rw tag-type?      dot1q-types:dot1q-tag-type
              +--:(dot1q-vlan-tagged)
                +--rw dot1q-vlan-tagged
                  +--rw outer-tag!
                    | +--rw dot1q-tag
                    |   +--rw tag-type    dot1q-tag-type
                    |   +--rw vlan-id     union
                  +--rw second-tag!
                    | +--rw dot1q-tag
                    |   +--rw tag-type    dot1q-tag-type
                    |   +--rw vlan-id     union
                  +--rw match-exact-tags? empty
          +--rw rewrite {flexible-rewrites}?

```

```

+--rw (direction)?
+--:(symmetrical)
|   +--rw symmetrical
|       +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|           +--rw pop-tags?      uint8
|           +--rw push-tags
|               +--rw outer-tag!
|                   +--rw dot1q-tag
|                       +--rw tag-type      dot1q-tag-type
|                       +--rw vlan-id      ieee:vlanid
|               +--rw second-tag!
|                   +--rw dot1q-tag
|                       +--rw tag-type      dot1q-tag-type
|                       +--rw vlan-id      ieee:vlanid
+--:(asymmetrical) {asymmetric-rewrites}?
+--rw ingress
|   +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|       +--rw pop-tags?      uint8
|       +--rw push-tags
|           +--rw outer-tag!
|               +--rw dot1q-tag
|                   +--rw tag-type      dot1q-tag-type
|                   +--rw vlan-id      ieee:vlanid
|       +--rw second-tag!
|           +--rw dot1q-tag
|               +--rw tag-type      dot1q-tag-type
|               +--rw vlan-id      ieee:vlanid
+--rw egress
|   +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|       +--rw pop-tags?      uint8
|       +--rw push-tags
|           +--rw outer-tag!
|               +--rw dot1q-tag
|                   +--rw tag-type      dot1q-tag-type
|                   +--rw vlan-id      ieee:vlanid
|       +--rw second-tag!
|           +--rw dot1q-tag
|               +--rw tag-type      dot1q-tag-type
|               +--rw vlan-id      ieee:vlanid
+--rw local-traffic-default-encaps!
+--rw outer-tag!
|   +--rw dot1q-tag
|       +--rw tag-type      dot1q-tag-type
|       +--rw vlan-id      ieee:vlanid
+--rw second-tag!
|   +--rw dot1q-tag
|       +--rw tag-type      dot1q-tag-type
|       +--rw vlan-id      ieee:vlanid

```

5. L3 Interface VLAN YANG Module

This YANG module augments the encapsulation container defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

```
<CODE BEGINS> file "ietf-if-l3-vlan@2017-07-03.yang"
module ietf-if-l3-vlan {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-if-l3-vlan";

  prefix if-l3-vlan;

  import ietf-interfaces {
    prefix if;
  }

  import iana-if-type {
    prefix ianaift;
  }

  import ieee802-dot1q-types {
    prefix dot1q-types;
  }

  import ietf-interfaces-common {
    prefix if-cmn;
  }

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

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

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

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

    Editor:    Robert Wilton
               <mailto:rwilton@cisco.com>";

  description
    "This YANG module models L3 VLAN sub-interfaces";
```

```
revision 2017-07-03 {
  description "Latest draft revision";

  reference
    "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-02";
}

/*
 * Add support for the 802.1Q VLAN encapsulation syntax on layer 3
 * terminated VLAN sub-interfaces.
 */
augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
  "if-cmn:encaps-type" {
  when
    "derived-from-or-self(..if:type,
                          'ianaift:ethernetCsmacd') or
     derived-from-or-self(..if:type,
                          'ianaift:ieee8023adLag') or
     derived-from-or-self(..if:type,
                          'if-cmn:ethSubInterface')" {
    description
      "Applies only to Ethernet-like interfaces and
       sub-interfaces";
  }

  description
    "Augment the generic interface encapsulation with an
     basic 802.1Q VLAN encapsulation for sub-interfaces.";

  /*
   * Matches a single VLAN Id, or pair of VLAN Ids to classify
   * traffic into an L3 service.
   */
  case dot1q-vlan {
    container dot1q-vlan {
      must
        'count(..../if-cmn:forwarding-mode) = 0 or ' +
        'derived-from-or-self(..../if-cmn:forwarding-mode,' +
        ' "if-cmn:layer-3-forwarding")' {
          error-message
            "If the interface forwarding-mode leaf is set then it
             must be set to an identity that derives from
             layer-3-forwarding";

          description
            "The forwarding-mode leaf on an interface can
             optionally be used to enforce consistency of
             configuration";
        }
      }
    }
  }
}
```


6. Flexible Encapsulation YANG Module

This YANG module augments the encapsulation container defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

This YANG module also augments the interface container defined in [RFC7223].

```
<CODE BEGINS> file "ietf-flexible-encapsulation@2017-07-03.yang"
module ietf-flexible-encapsulation {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-flexible-encapsulation";

  prefix flex;

  import ietf-interfaces {
    prefix if;
  }

  import iana-if-type {
    prefix ianaift;
  }

  import ietf-interfaces-common {
    prefix if-cmn;
  }

  import ieee802-dot1q-types {
    prefix dot1q-types;
  }

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

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

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

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

    Editor:    Robert Wilton
```

```
<mailto:rwilton@cisco.com>;

description
  "This YANG module describes interface configuration for flexible
  VLAN matches and rewrites.";

revision 2017-07-03 {
  description "Latest draft revision";

  reference
    "Internet-Draft draft-ietf-netmod-sub-intf-vlan-model-02";
}

feature flexible-rewrites {
  description
    "This feature indicates whether the network element supports
    specifying flexible rewrite operations";
}

feature asymmetric-rewrites {
  description
    "This feature indicates whether the network element supports
    specifying different rewrite operations for the ingress
    rewrite operation and egress rewrite operation.";
}

feature dot1q-tag-rewrites {
  description
    "This feature indicates whether the network element supports
    the flexible rewrite functionality specifying flexible 802.1Q
    tag rewrites";
}

/*
 * flexible-match grouping.
 *
 * This grouping represents a flexible match.
 *
 * The rules for a flexible match are:
 *   1. default, untagged, priority tag, or a stack of tags.
 *   - Each tag in the stack of tags matches:
 *     1. tag type (802.1Q or 802.1ad) +
 *     2. tag value:
 *       i. single tag
 *       ii. set of tag ranges/values.
 *       iii. "any" keyword
 */
grouping flexible-match {
```

```
description "Flexible match";
choice match-type {
  mandatory true;
  description "Provides a choice of how the frames may be
    matched";

  case default {
    description "Default match";
    leaf default {
      type empty;
      description
        "Default match.  Matches all traffic not matched to any
        other peer sub-interface by a more specific
        encapsulation.";
    } // leaf default
  } // case default

  case untagged {
    description "Match untagged Ethernet frames only";
    leaf untagged {
      type empty;
      description
        "Untagged match.  Matches all untagged traffic.";
    } // leaf untagged
  } // case untagged

  case dot1q-priority-tagged {
    description
      "Match 802.1Q priority tagged Ethernet frames only";

    container dot1q-priority-tagged {
      description "802.1Q priority tag match";
      leaf tag-type {
        type dot1q-types:dot1q-tag-type;
        description "The 802.1Q tag type of matched priority
          tagged packets";
      }
    }
  }

}

case dot1q-vlan-tagged {
  container dot1q-vlan-tagged {
    description "Matches VLAN tagged frames";

    container outer-tag {
      presence "The outermost VLAN tag exists";

      description
```

```
        "Classifies traffic using the outermost VLAN tag on the
        frame.";

    uses
        'dot1q-types:'+
        'dot1q-tag-ranges-or-any-classifier-grouping';
}

container second-tag {
    must
        ' ../outer-tag/dot1q-tag/tag-type = "s-vlan" and ' +
        'dot1q-tag/tag-type = "c-vlan"' {

        error-message
            "When matching two tags, the outermost tag must be
            specified and of S-VLAN type and the second
            outermost tag must be of C-VLAN tag type";

        description
            "For IEEE 802.1Q interoperability, when matching two
            tags, it is required that the outermost tag exists
            and is an S-VLAN, and the second outermost tag is a
            C-VLAN";
    }

    presence "The second outermost VLAN tag exists";

    description
        "Classifies traffic using the second outermost VLAN tag
        on the frame.";

    uses
        'dot1q-types:'+
        'dot1q-tag-ranges-or-any-classifier-grouping';
}

leaf match-exact-tags {
    type empty;
    description
        "If set, indicates that all 802.1Q VLAN tags in the
        Ethernet frame header must be explicitly matched, i.e.
        the EtherType following the matched tags must not be a
        802.1Q tag EtherType. If unset then extra 802.1Q VLAN
        tags are allowed.";
    }
}
} // encaps-type
```

```
}

/*
 * Grouping for tag-rewrite that can be expressed either
 * symmetrically, or in the ingress and/or egress directions
 * independently.
 */
grouping dot1q-tag-rewrite {
  description "Flexible rewrite";
  leaf pop-tags {
    type uint8 {
      range 1..2;
    }
    description "The number of tags to pop (or translate if used in
      conjunction with push-tags)";
  }

  container push-tags {
    description "The 802.1Q tags to push (or translate if used in
      conjunction with pop-tags)";
  }

  container outer-tag {
    presence
      "Indicates existence of the outermost VLAN tag to
      push/rewrite";

    description
      "The outermost VLAN tag to push onto the frame.";

    uses dot1q-types:dot1q-tag-classifier-grouping;
  }

  container second-tag {
    must
      '../outer-tag/dot1q-tag/tag-type = "s-vlan" and ' +
      'dot1q-tag/tag-type = "c-vlan"' {

      error-message
        "When pushing/rewriting two tags, the outermost tag must be
        specified and of S-VLAN type and the second outermost tag
        must be of C-VLAN tag type";

      description
        "For IEEE 802.1Q interoperability, when pushing two tags,
        it is required that the outermost tag exists and is an
        S-VLAN, and the second outermost tag is a C-VLAN";
    }
  }
}
```

```

        presence
            "Indicates existence of a second outermost VLAN tag to
            push/rewrite.";

        description
            "The second outermost VLAN tag to push onto the frame.";

        uses dot1q-types:dot1q-tag-classifier-grouping;
    }
}

/*
 * Grouping for all flexible rewrites of fields in the L2 header.
 *
 * This currently only includes flexible tag rewrites, but is
 * designed to be extensible to cover rewrites of other fields in
 * the L2 header if required.
 */
grouping flexible-rewrite {
    description "Flexible rewrite";

    /*
     * Tag rewrite.
     *
     * All tag rewrites are formed using a combination of pop-tags
     * and push-tags operations.
     */
    container dot1q-tag-rewrite {
        if-feature dot1q-tag-rewrites;
        description "Tag rewrite. Translate operations are expressed
            as a combination of tag push and pop operations.";
        uses dot1q-tag-rewrite;
    }
}

augment "/if:interfaces/if:interface/if-cmn:encapsulation/" +
    "if-cmn:encaps-type" {
    when
        "derived-from-or-self(..if:type,
            'ianaift:ethernetCsmacd') or
        derived-from-or-self(..if:type,
            'ianaift:ieee8023adLag') or
        derived-from-or-self(..if:type,
            'if-cmn:ethSubInterface')" {
        description
            "Applies only to Ethernet-like interfaces and
            sub-interfaces";
    }
}

```

```
description
  "Add flexible match and rewrite for VLAN sub-interfaces";

/*
 * A flexible encapsulation allows for the matching of ranges and
 * sets of VLAN Ids. The structure is also designed to be
 * extended to allow for matching/rewriting other fields within
 * the L2 frame header if required.
 */
case flexible {
  description "Flexible encapsulation and rewrite";
  container flexible {
    must
      'count(.../.../if-cmn:forwarding-mode) = 0 or ' +
      'derived-from-or-self(.../.../if-cmn:forwarding-mode,' +
      ' "if-cmn:layer-2-forwarding")' {
    error-message
      "If the interface forwarding-mode leaf is set then it
      must be set to an identity that derives from
      layer-2-forwarding";

    description
      "The forwarding-mode leaf on an interface can
      optionally be used to enforce consistency of
      configuration";
  }

  description "Flexible encapsulation and rewrite";

  container match {
    description
      "The match used to classify frames to this interface";
    uses flexible-match;
  }

  container rewrite {
    if-feature flexible-rewrites;
    description "L2 frame rewrite operations";
    choice direction {
      description
        "Whether the rewrite policy is symmetrical or
        asymmetrical";
      case symmetrical {
        container symmetrical {
          uses flexible-rewrite;
          description
            "Symmetrical rewrite. Expressed in the ingress
            direction, but the reverse operation is applied to
```

```
        egress traffic";
    }
}

/*
 * Allow asymmetrical rewrites to be specified.
 */
case asymmetrical {
    if-feature asymmetric-rewrites;
    description "Asymmetrical rewrite";
    container ingress {
        uses flexible-rewrite;
        description "Ingress rewrite";
    }
    container egress {
        uses flexible-rewrite;
        description "Egress rewrite";
    }
}
}

/*
 * For encapsulations that match a range of VLANs (or Any),
 * allow configuration to specify the default 802.1Q VLAN tag
 * values to use for any traffic that is locally sourced from
 * an interface on the device.
 */
container local-traffic-default-encaps {
    presence
        "A local traffic default encapsulation has been
        specified";
    description
        "The 802.1Q VLAN tags to use by default for locally
        sourced traffic";

    container outer-tag {
        presence
            "Indicates existence of the outermost VLAN tag";

        description
            "The outermost VLAN tag for locally sourced traffic";

        uses dot1q-types:dot1q-tag-classifier-grouping;
    }

    container second-tag {
        must
```



```
'../outer-tag/dot1q-tag/tag-type = "s-vlan" and ' +
'dot1q-tag/tag-type = "c-vlan"' {

error-message
    "When specifying two tags, the outermost tag must be
      specified and of S-VLAN type and the second outermost
      tag must be of C-VLAN tag type";

description
    "For IEEE 802.1Q interoperability, when specifying two
      tags, it is required that the outermost tag exists and
      is an S-VLAN, and the second outermost tag is a
      C-VLAN";
}

presence
    "Indicates existence of a second outermost VLAN tag.";

description
    "The second outermost VLAN tag for locally sourced
      traffic";

uses dot1q-types:dot1q-tag-classifier-grouping;
}
}
}
}
}
<CODE ENDS>
```

7. Open Issues

Open issues:

1. Consider whether to use interface property identities (as per draft-wilton-netmod-interface-properties).
2. Provide configuration examples?
3. Remove extra 'dot1q-tag' container (required update to IEEE YANG file).

8. Acknowledgements

The authors would particularly like to thank John Messenger, Glenn Parsons, and Dan Romascanu for their help progressing this draft.

The authors would also like to thank Alex Campbell, Eric Gray, Giles Heron, Marc Holness, Iftekhar Hussain, Neil Ketley, William Lupton, John Messenger, Glenn Parsons, Ludwig Pauwels, Joseph White, and members of the IEEE 802.1 WG for their helpful reviews and feedback on this draft.

9. ChangeLog

9.1. WG version -02

- o Use explicit containers for outer and inner tags rather than lists.

9.2. WG version -01

- o Tweaked the abstract.
- o Removed unnecessary feature for the L3 sub-interface module.
- o Update the 802.1Qcp type references.
- o Remove extra tag container for L3 sub-interfaces YANG.

9.3. Version -04

- o IEEE 802.1 specific types have been removed from the draft. These are now referenced from the 802.1Qcp draft YANG modules.
- o Fixed errors in the xpath expressions.

9.4. Version -03

- o Incorporates feedback received from presenting to the IEEE 802.1 WG.
- o Updates the modules for double tag matches/rewrites to restrict the outer tag type to S-VLAN and inner tag type to C-VLAN.
- o Updates the introduction to indicate primary use case is for IETF forwarding protocols.
- o Updates the objectives to make IEEE 802.1Q bridge interoperability a key objective.

10. IANA Considerations

This document defines several new YANG module and the authors politely request that IANA assigns unique names to the YANG module files contained within this draft, and also appropriate URIs in the "IETF XML Registry".

11. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [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 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:

11.1. if-l3-vlan.yang

The nodes in the if-l3-vlan YANG module are concerned with matching particular frames received on the network device to connect them to a layer 3 forwarding instance, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/dot1q-vlan, that are sensitive to this are:

- o outer-tag/dot1q-tag/tag-type
- o outer-tag/dot1q-tag/vlan-id
- o second-tag/dot1q-tag/tag-type
- o second-tag/dot1q-tag/vlan-id

11.2. flexible-encapsulation.yang

There are many nodes in the flexible-encapsulation YANG module that are concerned with matching particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree `/interfaces/interface/encapsulation/flexible/match`, that are sensitive to this are:

- o `default`
- o `untagged`
- o `dot1q-priority-tagged`
- o `dot1q-priority-tagged/tag-type`
- o `dot1q-vlan-tagged/outer-tag/dot1q-tag/vlan-type`
- o `dot1q-vlan-tagged/outer-tag/dot1q-tag/vlan-id`
- o `dot1q-vlan-tagged/second-tag/dot1q-tag/vlan-type`
- o `dot1q-vlan-tagged/second-tag/dot1q-tag/vlan-id`

There are also many nodes in the flexible-encapsulation YANG module that are concerned with rewriting the fields in the L2 header for particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be dropped or incorrectly processed on peer network devices, or it could cause layer 2 tunnels to go down due to a mismatch in negotiated MTU. The nodes, all under the subtree `/interfaces/interface/encapsulation/flexible/rewrite`, that are sensitive to this are:

- o `symmetrical/dot1q-tag-rewrite/pop-tags`
- o `symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/dot1q-tag/tag-type`
- o `symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/dot1q-tag/vlan-id`
- o `symmetrical/dot1q-tag-rewrite/push-tags/second-tag/dot1q-tag/tag-type`

- o symmetrical/dot1q-tag-rewrite/push-tags/second-tag/dot1q-tag/vlan-id
- o asymmetrical/ingress/dot1q-tag-rewrite/pop-tags
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/dot1q-tag/tag-type
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/dot1q-tag/vlan-id
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/dot1q-tag/tag-type
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/dot1q-tag/vlan-id
- o asymmetrical/egress/dot1q-tag-rewrite/pop-tags
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/dot1q-tag/tag-type
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/dot1q-tag/vlan-id
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/dot1q-tag/tag-type
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/dot1q-tag/vlan-id

Nodes in the flexible-encapsulation YANG module that are concerned with the VLAN tags to use for traffic sourced from the network element could cause protocol sessions (such as CFM) to fail if they are added, modified or deleted. The nodes, all under the subtree /interfaces/interface/flexible-encapsulation/local-traffic-default-encaps that are sensitive to this are:

- o outer-tag/dot1q-tag/vlan-type
- o outer-tag/dot1q-tag/vlan-id
- o second-tag/dot1q-tag/vlan-type
- o second-tag/dot1q-tag/vlan-id

12. References

12.1. Normative References

- [I-D.ietf-netmod-intf-ext-yang]
Wilton, R., Ball, D., tsingh@juniper.net, t., and S. Sivaraj, "Common Interface Extension YANG Data Models", draft-ietf-netmod-intf-ext-yang-04 (work in progress), March 2017.
- [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>>.
- [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>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<http://www.rfc-editor.org/info/rfc7224>>.
- [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

- [dot1Qcp] Holness, M., "802.1Qcp Bridges and Bridged Networks - Amendment: YANG Data Model", 2016.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <<http://www.rfc-editor.org/info/rfc2460>>.
- [RFC4448] Martini, L., Ed., Rosen, E., El-Aawar, N., and G. Heron, "Encapsulation Methods for Transport of Ethernet over MPLS Networks", RFC 4448, DOI 10.17487/RFC4448, April 2006, <<http://www.rfc-editor.org/info/rfc4448>>.
- [RFC4761] Kompella, K., Ed. and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <<http://www.rfc-editor.org/info/rfc4761>>.

- [RFC4762] Lasserre, M., Ed. and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <<http://www.rfc-editor.org/info/rfc4762>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

Appendix A. Comparison with the IEEE 802.1Q Configuration Model

In addition to the sub-interface based YANG model proposed here, the IEEE 802.1Q working group is also developing a YANG model for the configuration of 802.1Q VLANs. This raises the valid question as to whether the models overlap and whether it is necessary or beneficial to have two different models for superficially similar constructs. This section aims to answer that question by summarizing and comparing the two models.

A.1. Sub-interface based configuration model overview

The key features of the sub-interface based configuration model can be summarized as:

- o The model is primarily designed to enable layer 2 and layer 3 services on Ethernet interfaces that can be defined in a very flexible way to meet the varied requirements of service providers.
- o Traffic is classified from an Ethernet-like interface to sub-interfaces based on fields in the layer 2 header. This is often based on VLAN Ids contained in the frame, but the model is extensible to other arbitrary fields in the frame header.
- o Sub-interfaces are just a type of if:interface and hence support any feature configuration YANG models that can be applied generally to interfaces. For example, QoS or ACL models that reference if:interface can be applied to the sub-interfaces, or

the sub-interface can be used as an Access Circuit in L2VPN or L3VPN models that reference if:interface.

- o In the sub-interface based configuration model, the classification of traffic arriving on an interface to a given sub-interface, based on fields in the layer 2 header, is completely independent of how the traffic is forwarded. The sub-interface can be referenced (via references to if:interface) by other models that specify how traffic is forwarded; thus sub-interfaces can support multiple different forwarding paradigms, including but not limited to: layer 3 (IPv4/IPv6), layer 2 pseudowires (over MPLS or IP), VPLS instances, EVPN instance.
- o The model is flexible in the scope of the VLAN Identifier space. I.e. by default VLAN Ids can be scoped locally to a single Ethernet-like trunk interface, but the scope is determined by the forwarding paradigm that is used.

A.2. IEEE 802.1Q Bridge Configuration Model Overview

The key features of the IEEE 802.1Q bridge configuration model can be summarized as:

- o Each VLAN bridge component has a set of Ethernet interfaces that are members of that bridge. Sub-interfaces are not used, nor required in the 802.1Q bridge model.
- o Within a VLAN bridge component, the VLAN tag in the packet is used, along with the destination MAC address, to determine how to forward the packet. Other forwarding paradigms are not supported by the 802.1Q model.
- o Classification of traffic to a VLAN bridge component is based only on the Ethernet interface that it arrived on.
- o VLAN Identifiers are scoped to a VLAN bridge component. Often devices only support a single bridge component and hence VLANs are scoped globally within the device.
- o Feature configuration is specified in the context of the bridge, or particular VLANs on a bridge.

A.3. Possible Overlap Between the Two Models

Both models can be used for configuring similar basic layer 2 forwarding topologies. The 802.1Q bridge configuration model is optimised for configuring Virtual LANs that span across enterprises and data centers.

The sub-interface model can also be used for configuring equivalent Virtual LAN networks that span across enterprises and data centers, but often requires more configuration to be able to configure the equivalent constructs to the 802.1Q bridge model.

The sub-interface model really excels when implementing flexible L2 and L3 services, where those services may be handled on the same physical interface, and where the VLAN Identifier is being solely used to identify the customer or service that is being provided rather than a Virtual LAN. The sub-interface model provides more flexibility as to how traffic can be classified, how features can be applied to traffic streams, and how the traffic is to be forwarded.

Conversely, the 802.1Q bridge model can also be use to implement L2 services in some scenarios, but only if the forwarding paradigm being used to implement the service is the native Ethernet forwarding specified in 802.1Q - other forwarding paradigms such as pseudowires or VPLS are not supported. The 802.1Q bridge model does not implement L3 services at all, although this can be partly mitigated by using a virtual L3 interface construct that is a separate logical Ethernet-like interface which is a member of the bridge.

In conclusion, it is valid for both of these models to exist since they have different deployment scenarios for which they are optimized. Devices may choose which of the models (or both) to implement depending on what functionality the device is being designed for.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@cisco.com

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2021

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
July 13, 2020

Sub-interface VLAN YANG Data Models
draft-ietf-netmod-sub-intf-vlan-model-07

Abstract

This document defines YANG modules to add support for classifying traffic received on interfaces as Ethernet/VLAN framed packets to sub-interfaces based on the fields available in the Ethernet/VLAN frame headers. These modules allow configuration of Layer 3 and Layer 2 sub-interfaces (e.g. L2VPN attachment circuits) that can interoperate with IETF based forwarding protocols; such as IP and L3VPN services; or L2VPN services like VPWS, VPLS, and EVPN. The sub-interfaces also interoperate with VLAN tagged traffic originating from an IEEE 802.1Q compliant bridge.

The model differs from an IEEE 802.1Q bridge model in that the configuration is interface/sub-interface based as opposed to being based on membership of an 802.1Q VLAN bridge.

The YANG data models in this document conforms to the Network Management Datastore Architecture (NMDA) defined in RFC 8342.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 14, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Tree Diagrams	4
2. Objectives	4
2.1. Interoperability with IEEE 802.1Q compliant bridges	4
3. Interface VLAN Encapsulation Model	4
4. Interface Flexible Encapsulation Model	5
5. VLAN Encapsulation YANG Module	7
6. Flexible Encapsulation YANG Module	11
7. Examples	21
7.1. Layer 3 sub-interfaces with IPv6	22
7.2. Layer 2 sub-interfaces with L2VPN	23
8. Acknowledgements	26
9. ChangeLog	26
9.1. WG version -07 and -06	26
9.2. WG version -05	26
9.3. WG version -04	26
9.4. WG version -03	27
9.5. WG version -02	27
9.6. WG version -01	27
9.7. Version -04	27
9.8. Version -03	27
10. IANA Considerations	27
10.1. YANG Module Registrations	27
11. Security Considerations	28
11.1. ietf-if-vlan-encapsulation.yang	29
11.2. ietf-if-flexible-encapsulation.yang	29
12. References	31
12.1. Normative References	31
12.2. Informative References	32
Appendix A. Comparison with the IEEE 802.1Q Configuration Model	33

A.1. Sub-interface based configuration model overview	33
A.2. IEEE 802.1Q Bridge Configuration Model Overview	34
A.3. Possible Overlap Between the Two Models	34
Authors' Addresses	35

1. Introduction

This document defines two YANG [RFC7950] modules that augment the encapsulation choice YANG element defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang] and the generic interfaces data model defined in [RFC8343]. The two modules provide configuration nodes to support classification of Ethernet/VLAN traffic to sub-interfaces, that can have interface based feature and service configuration applied to them.

The purpose of these models is to allow IETF defined forwarding protocols, for example, IPv6 [RFC2460], Ethernet Pseudo Wires [RFC4448] and VPLS [RFC4761] [RFC4762], when configured via appropriate YANG data models [RFC8344] [I-D.ietf-bess-l2vpn-yang], to interoperate with VLAN tagged traffic received from an IEEE 802.1Q compliant bridge.

In the case of layer 2 Ethernet services, the flexible encapsulation module also supports flexible rewriting of the VLAN tags contained in the frame header.

For reference, a comparison between the sub-interface based YANG model documented in this draft and an IEEE 802.1Q bridge model is described in Appendix A.

In summary, the YANG modules defined in this internet draft are:

ietf-if-vlan-encapsulation.yang - Defines the model for basic classification of VLAN tagged traffic, normally to L3 packet forwarding services

ietf-if-flexible-encapsulation.yang - Defines the model for flexible classification of Ethernet/VLAN traffic, normally to L2 frame forwarding services

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 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

The term 'sub-interface' is defined in section 2.6 of Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

1.2. Tree Diagrams

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

2. Objectives

The primary aim of the YANG modules contained in this draft is to provide the core model that is required to implement VLAN transport services on router based devices that is fully compatible with IEEE 802.1Q compliant bridges.

A secondary aim is for the modules to be structured in such a way that they can be cleanly extended in future.

2.1. Interoperability with IEEE 802.1Q compliant bridges

The modules defined in this document are designed to fully interoperate with IEEE 802.1Q compliant bridges. In particular, the models are restricted to only matching, adding, or rewriting the 802.1Q VLAN tags in frames in ways that are compatible with IEEE 802.1Q compliant bridges.

3. Interface VLAN Encapsulation Model

The Interface VLAN encapsulation model provides appropriate leaves for termination of an 802.1Q VLAN tagged segment to a sub-interface (or interface) based L3 service, such as IP. It allows for termination of traffic with one or two 802.1Q VLAN tags.

The L3 service must be configured via a separate YANG data model, e.g., [RFC8344]. A short example of configuring 802.1Q VLAN sub-interfaces with IP using YANG is provided in Section 7.1.

The "ietf-if-vlan-encapsulation" YANG module has the following structure:

```

module: ietf-if-vlan-encapsulation
  augment /if:interfaces/if:interface/if-ext:encapsulation
    /if-ext:encaps-type:
      +--:(dot1q-vlan)
        +--rw dot1q-vlan
          +--rw outer-tag
            +--rw tag-type      dot1q-tag-type
            +--rw vlan-id       vlanid
          +--rw second-tag!
            +--rw tag-type      dot1q-tag-type
            +--rw vlan-id       vlanid

```

4. Interface Flexible Encapsulation Model

The Interface Flexible Encapsulation model is designed to allow for the flexible provisioning of layer 2 services. It provides the capability to classify and demultiplex Ethernet/VLAN frames received on an Ethernet trunk interface to sub-interfaces based on the fields available in the layer 2 headers. Once classified to sub-interfaces, it provides the capability to selectively modify fields within the layer 2 frame header before the frame is handed off to the appropriate forwarding code for further handling. The forwarding instance, e.g., L2VPN, VPLS, etc., is configured using a separate YANG configuration model defined elsewhere, e.g., [I-D.ietf-bess-l2vpn-yang].

The model supports a common core set of layer 2 header matches based on the 802.1Q tag type and VLAN Ids contained within the header up to a tag stack depth of two tags.

The model supports flexible rewrites of the layer 2 frame header for data frames as they are processed on the interface. It defines a set of standard tag manipulations that allow for the insertion, removal, or rewrite of one or two 802.1Q VLAN tags. The expectation is that manipulations are generally implemented in a symmetrical fashion, i.e. if a manipulation is performed on ingress traffic on an interface then the reverse manipulation is always performed on egress traffic out of the same interface. However, the model also allows for asymmetrical rewrites, which may be required to implement some forwarding models (such as E-Tree).

The model also allows a flexible encapsulation and rewrite to be configured directly on an Ethernet or LAG interface without

configuring separate child sub-interfaces. Ingress frames that do not match the encapsulation are dropped. Egress frames MUST conform to the encapsulation.

The final aim for the model design is for it to be cleanly extensible to add in additional match and rewrite criteria of the layer 2 header, such as matching on the source or destination MAC address, PCP or DEI fields in the 802.1Q tags, or the EtherType of the frame payload. Rewrites can also be extended to allow for modification of other fields within the layer 2 frame header.

A short example of configuring 802.1Q VLAN sub-interfaces with L2VPN using YANG is provided in Section 7.2.

The "ietf-if-flexible-encapsulation" YANG module has the following structure:

```

module: ietf-if-flexible-encapsulation
  augment /if:interfaces/if:interface/if-ext:encapsulation
    /if-ext:encaps-type:
      +--:(flexible)
        +--rw flexible
          +--rw match
            +--rw (match-type)
              +--:(default)
                | +--rw default?          empty
              +--:(untagged)
                | +--rw untagged?          empty
              +--:(dot1q-priority-tagged)
                | +--rw dot1q-priority-tagged
                |   +--rw tag-type      dot1q-types:dot1q-tag-type
              +--:(dot1q-vlan-tagged)
                +--rw dot1q-vlan-tagged
                  +--rw outer-tag
                    +--rw tag-type      dot1q-tag-type
                    +--rw vlan-id        union
                  +--rw second-tag!
                    +--rw tag-type      dot1q-tag-type
                    +--rw vlan-id        union
                  +--rw match-exact-tags? empty
          +--rw rewrite {flexible-rewrites}?
            +--rw (direction)?
              +--:(symmetrical)
                +--rw symmetrical
                  +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
                    +--rw pop-tags?      uint8
                    +--rw push-tags!

```



```

    +--rw outer-tag
    |   +--rw tag-type      dot1q-tag-type
    |   +--rw vlan-id      vlanid
    +--rw second-tag!
    |   +--rw tag-type      dot1q-tag-type
    |   +--rw vlan-id      vlanid
+--:(asymmetrical) {asymmetric-rewrites}?
+--rw ingress
|   +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|   +--rw pop-tags?      uint8
|   +--rw push-tags!
|   |   +--rw outer-tag
|   |   |   +--rw tag-type      dot1q-tag-type
|   |   |   +--rw vlan-id      vlanid
|   |   +--rw second-tag!
|   |   |   +--rw tag-type      dot1q-tag-type
|   |   |   +--rw vlan-id      vlanid
+--rw egress
|   +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|   +--rw pop-tags?      uint8
|   +--rw push-tags!
|   |   +--rw outer-tag
|   |   |   +--rw tag-type      dot1q-tag-type
|   |   |   +--rw vlan-id      vlanid
|   |   +--rw second-tag!
|   |   |   +--rw tag-type      dot1q-tag-type
|   |   |   +--rw vlan-id      vlanid
+--rw local-traffic-default-encaps!
+--rw outer-tag
|   +--rw tag-type      dot1q-tag-type
|   +--rw vlan-id      vlanid
+--rw second-tag!
|   +--rw tag-type      dot1q-tag-type
|   +--rw vlan-id      vlanid

```

5. VLAN Encapsulation YANG Module

This YANG module augments the 'encapsulation' container defined in `ietf-if-extensions.yang` [I-D.ietf-netmod-intf-ext-yang]. It also contains references to [RFC8343], [RFC7224], and [IEEE802.1Qcp-2018].

```

<CODE BEGINS> file "ietf-if-vlan-encapsulation@2020-07-13.yang"
module ietf-if-vlan-encapsulation {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation";
  prefix if-vlan;

```

```
import ietf-interfaces {
  prefix if;
  reference
    "RFC 8343: A YANG Data Model For Interface Management";
}

import iana-if-type {
  prefix ianaift;
  reference
    "RFC 7224: IANA Interface Type YANG Module";
}

import ieee802-dot1q-types {
  prefix dot1q-types;
  reference
    "IEEE Std 802.1Qcp-2018: IEEE Standard for Local and
    metropolitan area networks -- Bridges and Bridged Networks --
    Amendment 30: YANG Data Model";
}

import ietf-if-extensions {
  prefix if-ext;
  reference
    "RFC XXXX: Common Interface Extension YANG Data Models";
}

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

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

  Editor:   Robert Wilton
            <mailto:rwilton@cisco.com>";

description
  "This YANG module models configuration to classify IEEE 802.1Q
  VLAN tagged Ethernet traffic by exactly matching the tag type
  and VLAN identifier of one or two 802.1Q VLAN tags in the frame.

  Copyright (c) 2020 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>).

This version of this YANG module is part of RFC XXXX
(<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself
for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

```
revision 2020-07-13 {
  description
    "Latest draft revision";
  reference
    "RFC XXXX: Sub-interface VLAN YANG Data Models";
}

augment "/if:interfaces/if:interface/if-ext:encapsulation/"
+ "if-ext:encaps-type" {
  when "derived-from-or-self(..if:type,
    'ianaift:ethernetCsmacd') or
    derived-from-or-self(..if:type,
    'ianaift:ieee8023adLag') or
    derived-from-or-self(..if:type, 'ianaift:l2vlan') or
    derived-from-or-self(..if:type,
    'if-ext:ethSubInterface')" {
    description
      "Applies only to Ethernet-like interfaces and
      sub-interfaces.";
  }
}

description
  "Augment the generic interface encapsulation with basic 802.1Q
  VLAN tag classifications";

case dot1q-vlan {
  container dot1q-vlan {

    description
      "Classifies 802.1Q VLAN tagged Ethernet frames to a
      sub-interface (or interface) by exactly matching the
      number of tags, tag type(s) and VLAN identifier(s).

      Only frames matching the classification configured on a
      sub-interface/interface are processed on that
```

sub-interface/interface.

Frames that do not match any sub-interface are processed directly on the parent interface, if it is associated with a forwarding instance, otherwise they are dropped.";

```
container outer-tag {
  must 'tag-type = "dot1q-types:s-vlan" or '
    + 'tag-type = "dot1q-types:c-vlan"' {

    error-message
      "Only C-VLAN and S-VLAN tags can be matched.";

    description
      "For IEEE 802.1Q interoperability, only C-VLAN and
        S-VLAN tags are matched.";
  }

  description
    "Specifies the VLAN tag values to match against the
      outermost (first) 802.1Q VLAN tag in the frame.";

  uses dot1q-types:dot1q-tag-classifier-grouping;
}

container second-tag {
  must '../outer-tag/tag-type = "dot1q-types:s-vlan" and '
    + 'tag-type = "dot1q-types:c-vlan"' {

    error-message
      "When matching two 802.1Q VLAN tags, the outermost
        (first) tag in the frame MUST be specified and be of
        S-VLAN type and the second tag in the frame must be of
        C-VLAN tag type.";

    description
      "For IEEE 802.1Q interoperability, when matching two
        802.1Q VLAN tags, it is REQUIRED that the outermost
        tag exists and is an S-VLAN, and the second tag is a
        C-VLAN.";
  }

  presence "Classify frames that have two 802.1Q VLAN tags.";

  description
    "Specifies the VLAN tag values to match against the
      second outermost 802.1Q VLAN tag in the frame.";
```

```
        uses dot1q-types:dot1q-tag-classifier-grouping;
    }
}
}
}
}
<CODE ENDS>
```

6. Flexible Encapsulation YANG Module

This YANG module augments the 'encapsulation' container defined in ietf-if-extensions.yang [I-D.ietf-netmod-intf-ext-yang]. This YANG module also augments the 'interface' list entry defined in [RFC8343]. It also contains references to [RFC7224], and [IEEE802.1Qcp-2018].

```
<CODE BEGINS> file "ietf-if-flexible-encapsulation@2020-07-13.yang"
module ietf-if-flexible-encapsulation {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation";
  prefix if-flex;

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

  import iana-if-type {
    prefix ianaift;
    reference
      "RFC 7224: IANA Interface Type YANG Module";
  }

  import ieee802-dot1q-types {
    prefix dot1q-types;
    reference
      "IEEE Std 802.1Qcp-2018: IEEE Standard for Local and
      metropolitan area networks -- Bridges and Bridged Networks --
      Amendment 30: YANG Data Model";
  }

  import ietf-if-extensions {
    prefix if-ext;
    reference
      "RFC XXXX: Common Interface Extension YANG Data Models";
```

```
}

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

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

  Editor:   Robert Wilton
            <mailto:rwilton@cisco.com>";

description
  "This YANG module describes interface configuration for flexible
  classification and rewrites of IEEE 802.1Q VLAN tagged Ethernet
  traffic.

  Copyright (c) 2020 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).

  This version of this YANG module is part of RFC XXXX
  (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
  for full legal notices.

  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 (RFC 2119) (RFC 8174) when, and only when,
  they appear in all capitals, as shown here.";

revision 2020-07-13 {
  description
    "Latest draft revision";
  reference
    "RFC XXXX: Sub-interface VLAN YANG Data Models";
}

feature flexible-rewrites {
  description
    "This feature indicates that the network element supports
    specifying flexible rewrite operations.";
```

```
}

feature asymmetric-rewrites {
  description
    "This feature indicates that the network element supports
    specifying different rewrite operations for the ingress
    rewrite operation and egress rewrite operation.";
}

feature dot1q-tag-rewrites {
  description
    "This feature indicates that the network element supports the
    flexible rewrite functionality specifying 802.1Q tag
    rewrites.";
}

grouping flexible-match {
  description
    "Represents a flexible frame classification:

    The rules for a flexible match are:
    1. Match-type: default, untagged, priority tag, or tag
       stack.
    2. Each tag in the stack of tags matches:
       a. tag type (802.1Q or 802.1ad) +
       b. tag value:
          i. single tag
          ii. set of tag ranges/values.
          iii. 'any' keyword";

  choice match-type {
    mandatory true;

    description
      "Provides a choice of how the frames may be
      matched";

    case default {
      description
        "Default match";

      leaf default {
        type empty;

        description
          "Default match. Matches all traffic not matched to any
          other peer sub-interface by a more specific
          encapsulation.";
```

```
    }  
  }  
  
  case untagged {  
    description  
      "Match untagged Ethernet frames only";  
  
    leaf untagged {  
      type empty;  
  
      description  
        "Untagged match.  Matches all untagged traffic.";  
    }  
  }  
  
  case dot1q-priority-tagged {  
    description  
      "Match 802.1Q priority tagged Ethernet frames only";  
  
    container dot1q-priority-tagged {  
      description  
        "802.1Q priority tag match";  
  
      leaf tag-type {  
        type dot1q-types:dot1q-tag-type;  
        mandatory true;  
  
        description  
          "The 802.1Q tag type of matched priority  
          tagged packets";  
      }  
    }  
  }  
  
  case dot1q-vlan-tagged {  
    container dot1q-vlan-tagged {  
      description  
        "Matches VLAN tagged frames";  
  
      container outer-tag {  
        must 'tag-type = "dot1q-types:s-vlan" or '  
          + 'tag-type = "dot1q-types:c-vlan"' {  
  
          error-message  
            "Only C-VLAN and S-VLAN tags can be matched.";  
  
          description  
            "For IEEE 802.1Q interoperability, only C-VLAN and
```



```
        S-VLAN tags can be matched.";
    }

    description
        "Classifies traffic using the outermost (first) VLAN
        tag on the frame.";

    uses "dot1q-types:"
        + "dot1q-tag-ranges-or-any-classifier-grouping";
}

container second-tag {
    must
        ' ../outer-tag/tag-type = "dot1q-types:s-vlan" and '
        + 'tag-type = "dot1q-types:c-vlan"' {

        error-message
            "When matching two tags, the outermost (first) tag
            must be specified and of S-VLAN type and the second
            outermost tag must be of C-VLAN tag type.";

        description
            "For IEEE 802.1Q interoperability, when matching two
            tags, it is required that the outermost (first) tag
            exists and is an S-VLAN, and the second outermost
            tag is a C-VLAN.";
    }

    presence "Also classify on the second VLAN tag.";

    description
        "Classifies traffic using the second outermost VLAN tag
        on the frame.";

    uses "dot1q-types:"
        + "dot1q-tag-ranges-or-any-classifier-grouping";
}

leaf match-exact-tags {
    type empty;
    description
        "If set, indicates that all 802.1Q VLAN tags in the
        Ethernet frame header must be explicitly matched, i.e.
        the EtherType following the matched tags must not be a
        802.1Q tag EtherType. If unset then extra 802.1Q VLAN
        tags are allowed.";
}
}
```

```
    }  
  }  
}  
  
grouping dot1q-tag-rewrite {  
  description  
    "Flexible rewrite grouping. Can be either be expressed  
    symmetrically, or independently in the ingress and/or egress  
    directions.";  
  
  leaf pop-tags {  
    type uint8 {  
      range "1..2";  
    }  
  
    description  
      "The number of 802.1Q VLAN tags to pop, or translate if used  
      in conjunction with push-tags.  
  
      Popped tags are the outermost tags on the frame.";  
  }  
  
  container push-tags {  
    presence "802.1Q tags are pushed or translated";  
  
    description  
      "The 802.1Q tags to push on the front of the frame, or  
      translate if configured in conjunction with pop-tags.";  
  
    container outer-tag {  
      must 'tag-type = "dot1q-types:s-vlan" or '  
        + 'tag-type = "dot1q-types:c-vlan"' {  
  
        error-message "Only C-VLAN and S-VLAN tags can be pushed.";  
  
        description  
          "For IEEE 802.1Q interoperability, only C-VLAN and S-VLAN  
          tags can be pushed.";  
      }  
  
      description  
        "The outermost (first) VLAN tag to push onto the frame.";  
  
      uses dot1q-types:dot1q-tag-classifier-grouping;  
    }  
  
    container second-tag {  
      must '../outer-tag/tag-type = "dot1q-types:s-vlan" and '
```

```
    + 'tag-type = "dot1q-types:c-vlan"' {  
      error-message  
        "When pushing/rewriting two tags, the outermost tag must  
        be specified and of S-VLAN type and the second outermost  
        tag must be of C-VLAN tag type.";   
      description  
        "For IEEE 802.1Q interoperability, when pushing two tags,  
        it is required that the outermost tag exists and is an  
        S-VLAN, and the second outermost tag is a C-VLAN.";   
    }  
  presence  
    "In addition to the first tag, also push/rewrite a second  
    VLAN tag.";   
  description  
    "The second outermost VLAN tag to push onto the frame.";   
  uses dot1q-types:dot1q-tag-classifier-grouping;  
}  
}  
}  
grouping flexible-rewrite {  
  description  
    "Grouping for flexible rewrites of fields in the L2 header.  
  
    Restricted to flexible 802.1Q VLAN tag rewrites, but could be  
    extended to cover rewrites of other fields in the L2 header in  
    future.";   
  container dot1q-tag-rewrite {  
    if-feature "dot1q-tag-rewrites";  
    description  
      "802.1Q VLAN tag rewrite.  
  
      Translate operations are expressed as a combination of tag  
      push and pop operations.  E.g., translating the outer tag is  
      expressed as popping a single tag, and pushing a single tag.  
      802.1Q tags that are translated SHOULD preserve the PCP and  
      DEI fields unless if a different QoS behavior has been  
      specified.";   
    uses dot1q-tag-rewrite;  
  }  
}
```

```
augment "/if:interfaces/if:interface/if-ext:encapsulation/"
+ "if-ext:encaps-type" {
  when "derived-from-or-self(..if:type,
    'ianaift:ethernetCsmacd') or
    derived-from-or-self(..if:type,
    'ianaift:ieee8023adLag') or
    derived-from-or-self(..if:type, 'ianaift:l2vlan') or
    derived-from-or-self(..if:type,
    'if-ext:ethSubInterface')" {

    description
      "Applies only to Ethernet-like interfaces and
      sub-interfaces.";
  }

  description
    "Augment the generic interface encapsulation with flexible
    match and rewrite for VLAN sub-interfaces.";

  case flexible {
    description
      "Flexible encapsulation and rewrite";

    container flexible {
      description
        "Flexible encapsulation allows for the matching of ranges
        and sets of 802.1Q VLAN Tags and performing rewrite
        operations on the VLAN tags.

        The structure is also designed to be extended to allow for
        matching/rewriting other fields within the L2 frame header
        if required.";

      container match {
        description
          "Flexibly classifies Ethernet frames to a sub-interface
          (or interface) based on the L2 header fields.

          Only frames matching the classification configured on a
          sub-interface/interface are processed on that
          sub-interface/interface.

          Frames that do not match any sub-interface are processed
          directly on the parent interface, if it is associated
          with a forwarding instance, otherwise they are dropped.

          If a frame could be classified to multiple
          sub-interfaces then they get classified to the
```

```
        sub-interface with the most specific match.  E.g.,
        matching two VLAN tags in the frame is more specific
        than matching the outermost VLAN tag, which is more
        specific than the catch all 'default' match.";

    uses flexible-match;
}

container rewrite {
    if-feature "flexible-rewrites";

    description
        "L2 frame rewrite operations.

        Rewrites allows for modifications to the L2 frame header
        as it transits the interface/sub-interface.  Examples
        include adding a VLAN tag, removing a VLAN tag, or
        rewriting the VLAN Id carried in a VLAN tag.";

    choice direction {
        description
            "Whether the rewrite policy is symmetrical or
            asymmetrical.";

        case symmetrical {
            container symmetrical {
                uses flexible-rewrite;

                description
                    "Symmetrical rewrite.  Expressed in the ingress
                    direction, but the reverse operation is applied to
                    egress traffic.

                    E.g., if a tag is pushed on ingress traffic, then
                    the reverse operation is a 'pop 1', that is
                    performed on traffic egressing the interface, so
                    a peer device sees a consistent L2 encapsulation
                    for both ingress and egress traffic.";
            }
        }

        case asymmetrical {
            if-feature "asymmetric-rewrites";

            description
                "Asymmetrical rewrite.

                Rewrite operations may be specified in only a single
```

```
direction, or different rewrite operations may be
specified in each direction.";

container ingress {
  uses flexible-rewrite;

  description
    "A rewrite operation that only applies to ingress
    traffic.

    Ingress rewrite operations are performed before
    the frame is subsequently processed by the
    forwarding operation.";
}

container egress {
  uses flexible-rewrite;

  description
    "A rewrite operation that only applies to egress
    traffic.";
}
}

container local-traffic-default-encaps {
  presence "A local traffic default encapsulation has been
  specified.";

  description
    "Specifies the 802.1Q VLAN tags to use by default for
    locally sourced traffic from the interface.

    Used for encapsulations that match a range of VLANs (or
    'any'), where the source VLAN Ids are otherwise
    ambiguous.";

  container outer-tag {
    must 'tag-type = "dot1q-types:s-vlan" or '
      + 'tag-type = "dot1q-types:c-vlan"' {

      error-message
        "Only C-VLAN and S-VLAN tags can be matched.";

      description
        "For IEEE 802.1Q interoperability, only C-VLAN and
        S-VLAN tags can be matched.";
```


conjunction with the IETF L2VPN YANG model [I-D.ietf-bess-l2vpn-yang].

7.1. Layer 3 sub-interfaces with IPv6

This example illustrates two layer sub-interfaces, 'eth0.1' and 'eth0.2', both are child interfaces of the Ethernet interface 'eth0'.

'eth0.1' is configured to match traffic with two VLAN tags: an outer S-VLAN of 10 and an inner C-VLAN of 20.

'eth0.2' is configured to match traffic with a single S-VLAN tag, with VLAN Id 11.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
    xmlns:dot1q-types="urn:ieee:std:802.1Q:yang:ieee802-dot1q-types"
    xmlns:if-ext="urn:ietf:params:xml:ns:yang:ietf-if-extensions">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
    </interface>
    <interface>
      <name>eth0.1</name>
      <type>ianaift:l2vlan</type>
      <if-ext:parent-interface>eth0</if-ext:parent-interface>
      <if-ext:encapsulation>
        <dot1q-vlan
          xmlns=
            "urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation">
          <outer-tag>
            <tag-type>dot1q-types:s-vlan</tag-type>
            <vlan-id>10</vlan-id>
          </outer-tag>
          <second-tag>
            <tag-type>dot1q-types:c-vlan</tag-type>
            <vlan-id>20</vlan-id>
          </second-tag>
        </dot1q-vlan>
      </if-ext:encapsulation>
      <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
        <forwarding>true</forwarding>
        <address>
          <ip>2001:db8:10::1</ip>
```



```

        <prefix-length>48</prefix-length>
      </address>
    </ipv6>
  </interface>
  <interface>
    <name>eth0.2</name>
    <type>ianaift:l2vlan</type>
    <if-ext:parent-interface>eth0</if-ext:parent-interface>
    <if-ext:encapsulation>
      <dot1q-vlan
        xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation">
        <outer-tag>
          <tag-type>dot1q-types:s-vlan</tag-type>
          <vlan-id>11</vlan-id>
        </outer-tag>
      </dot1q-vlan>
    </if-ext:encapsulation>
    <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
      <forwarding>true</forwarding>
      <address>
        <ip>2001:db8:11::1</ip>
        <prefix-length>48</prefix-length>
      </address>
    </ipv6>
  </interface>
</interfaces>
</config>

```

7.2. Layer 2 sub-interfaces with L2VPN

This example illustrates a layer 2 sub-interface 'eth0.3' configured to match traffic with a S-VLAN tag of 10, and C-VLAN tag of 21; and remove the outer tag (S-VLAN 10) before the traffic is passed off to the L2VPN service.

It also illustrates another sub-interface 'eth1.0' under a separate physical interface configured to match traffic with a C-VLAN of 50, with the tag removed before traffic is given to any service. Sub-interface 'eth1.0' is not currently bound to any service and hence traffic classified to that sub-interface is dropped.

```

<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
    xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"

```

```
xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
xmlns:dot1q-types="urn:ieee:std:802.1Q:yang:ieee802-dot1q-types"
xmlns:if-ext="urn:ietf:params:xml:ns:yang:ietf-if-extensions">
  <interface>
    <name>eth0</name>
    <type>ianaift:ethernetCsmacd</type>
  </interface>
  <interface>
    <name>eth0.3</name>
    <type>ianaift:l2vlan</type>
    <if-ext:parent-interface>eth0</if-ext:parent-interface>
    <if-ext:encapsulation>
      <flexible xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation">
        <match>
          <dot1q-vlan-tagged>
            <outer-tag>
              <tag-type>dot1q-types:s-vlan</tag-type>
              <vlan-id>10</vlan-id>
            </outer-tag>
            <second-tag>
              <tag-type>dot1q-types:c-vlan</tag-type>
              <vlan-id>21</vlan-id>
            </second-tag>
          </dot1q-vlan-tagged>
        </match>
        <rewrite>
          <symmetrical>
            <dot1q-tag-rewrite>
              <pop-tags>1</pop-tags>
            </dot1q-tag-rewrite>
          </symmetrical>
        </rewrite>
      </flexible>
    </if-ext:encapsulation>
  </interface>
  <interface>
    <name>eth1</name>
    <type>ianaift:ethernetCsmacd</type>
  </interface>
  <interface>
    <name>eth1.0</name>
    <type>ianaift:l2vlan</type>
    <if-ext:parent-interface>eth0</if-ext:parent-interface>
    <if-ext:encapsulation>
      <flexible xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation">
        <match>
```

```
        <dot1q-vlan-tagged>
          <outer-tag>
            <tag-type>dot1q-types:c-vlan</tag-type>
            <vlan-id>50</vlan-id>
          </outer-tag>
        </dot1q-vlan-tagged>
      </match>
    <rewrite>
      <symmetrical>
        <dot1q-tag-rewrite>
          <pop-tags>1</pop-tags>
        </dot1q-tag-rewrite>
      </symmetrical>
    </rewrite>
  </flexible>
</if-ext:encapsulation>
</interface>
</interfaces>
<network-instances
  xmlns="urn:ietf:params:xml:ns:yang:ietf-network-instance">
  <network-instance
    xmlns:l2vpn="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
    <name>p2p-l2-1</name>
    <description>Point to point L2 service</description>
    <l2vpn:type>l2vpn:vpws-instance-type</l2vpn:type>
    <l2vpn:signaling-type>
      l2vpn:ldp-signaling
    </l2vpn:signaling-type>
    <endpoint xmlns="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
      <name>local</name>
      <ac>
        <name>eth0.3</name>
      </ac>
    </endpoint>
    <endpoint xmlns="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
      <name>remote</name>
      <pw>
        <name>pw1</name>
      </pw>
    </endpoint>
    <vsi-root>
    </vsi-root>
  </network-instance>
</network-instances>
<pseudowires
  xmlns="urn:ietf:params:xml:ns:yang:ietf-pseudowires">
  <pseudowire>
    <name>pw1</name>
```

```
<configured-pw>
  <peer-ip>2001:db8::50</peer-ip>
  <pw-id>100</pw-id>
</configured-pw>
</pseudowire>
</pseudowires>
</config>
```

8. Acknowledgements

The authors would particularly like to thank Benoit Claise, John Messenger, Glenn Parsons, and Dan Romascanu for their help progressing this draft.

The authors would also like to thank Martin Bjorklund, Alex Campbell, Don Fedyk, Eric Gray, Giles Heron, Marc Holness, Iftexhar Hussain, Neil Ketley, William Lupton, John Messenger, Glenn Parsons, Ludwig Pauwels, Joseph White, Vladimir Vassilev, and members of the IEEE 802.1 WG for their helpful reviews and feedback on this draft.

9. ChangeLog

XXX, RFC Editor, please delete this change log before publication.

9.1. WG version -07 and -06

- o Apply markups from WG last call.

9.2. WG version -05

- o Incorporate feedback from IEEE 802.1 WG, John Messenger in particular.
- o Adding must constraints to ensure outer tags are always matched to C-VLAN and S-VLAN tags.
- o Fixed bug where second tag could be matched without outer tag, and where tags must not be specified.

9.3. WG version -04

- o Added examples

9.4. WG version -03

- o Fix namespace bug in XPath identity references, removed extraneous 'dot1q-tag' containers.

9.5. WG version -02

- o Use explicit containers for outer and inner tags rather than lists.

9.6. WG version -01

- o Tweaked the abstract.
- o Removed unnecessary feature for the L3 sub-interface module.
- o Update the 802.1Qcp type references.
- o Remove extra tag container for L3 sub-interfaces YANG.

9.7. Version -04

- o IEEE 802.1 specific types have been removed from the draft. These are now referenced from the 802.1Qcp draft YANG modules.
- o Fixed errors in the xpath expressions.

9.8. Version -03

- o Incorporates feedback received from presenting to the IEEE 802.1 WG.
- o Updates the modules for double tag matches/rewrites to restrict the outer tag type to S-VLAN and inner tag type to C-VLAN.
- o Updates the introduction to indicate primary use case is for IETF forwarding protocols.
- o Updates the objectives to make IEEE 802.1Q bridge interoperability a key objective.

10. IANA Considerations

10.1. YANG Module Registrations

The following YANG modules are requested to be registered in the IANA "YANG Module Names" [RFC6020] registry:

The ietf-if-vlan-encapsulation module:

Name: ietf-if-vlan-encapsulation

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation

Prefix: if-vlan

Reference: [RFCXXXX]

The ietf-if-flexible-encapsulation module:

Name: ietf-if-flexible-encapsulation

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation

Prefix: if-flex

Reference: [RFCXXXX]

This document registers two URIs in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registrations have been made.

URI: urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation

Registrant Contact: The IESG.

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

11. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [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 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:

11.1. ietf-if-vlan-encapsulation.yang

The nodes in the vlan encapsulation YANG module are concerned with matching particular frames received on the network device to connect them to a layer 3 forwarding instance, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/dot1q-vlan, that are sensitive to this are:

- o outer-tag/tag-type
- o outer-tag/vlan-id
- o second-tag/tag-type
- o second-tag/vlan-id

11.2. ietf-if-flexible-encapsulation.yang

There are many nodes in the flexible encapsulation YANG module that are concerned with matching particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/match, that are sensitive to this are:

- o default
- o untagged
- o dot1q-priority-tagged
- o dot1q-priority-tagged/tag-type
- o dot1q-vlan-tagged/outer-tag/vlan-type

- o dot1q-vlan-tagged/outer-tag/vlan-id
- o dot1q-vlan-tagged/second-tag/vlan-type
- o dot1q-vlan-tagged/second-tag/vlan-id

There are also many modes in the flexible encapsulation YANG module that are concerned with rewriting the fields in the L2 header for particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be dropped or incorrectly processed on peer network devices, or it could cause layer 2 tunnels to go down due to a mismatch in negotiated MTU. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/rewrite, that are sensitive to this are:

- o symmetrical/dot1q-tag-rewrite/pop-tags
- o symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- o symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- o symmetrical/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- o symmetrical/dot1q-tag-rewrite/push-tags/second-tag/vlan-id
- o asymmetrical/ingress/dot1q-tag-rewrite/pop-tags
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/tag-type
- o asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/vlan-id
- o asymmetrical/egress/dot1q-tag-rewrite/pop-tags
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id
- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/tag-type

- o asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/vlan-id

Nodes in the flexible-encapsulation YANG module that are concerned with the VLAN tags to use for traffic sourced from the network element could cause protocol sessions (such as CFM) to fail if they are added, modified or deleted. The nodes, all under the subtree /interfaces/interface/flexible-encapsulation/local-traffic-default-encaps that are sensitive to this are:

- o outer-tag/vlan-type
- o outer-tag/vlan-id
- o second-tag/vlan-type
- o second-tag/vlan-id

12. References

12.1. Normative References

- [I-D.ietf-netmod-intf-ext-yang]
Wilton, R., Ball, D., tapsingh@cisco.com, t., and S. Sivaraj, "Common Interface Extension YANG Data Models", draft-ietf-netmod-intf-ext-yang-08 (work in progress), November 2019.
- [IEEE802.1Qcp-2018]
Holness, M., "IEEE Std 802.1Qcp-2018: IEEE Standard for Local and metropolitan area networks -- Bridges and Bridged Networks -- Amendment 30: YANG Data Model", 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>>.
- [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>>.

- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<https://www.rfc-editor.org/info/rfc7224>>.
- [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>>.
- [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>>.
- [RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <<https://www.rfc-editor.org/info/rfc8344>>.

12.2. Informative References

- [I-D.ietf-bess-l2vpn-yang] Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", draft-ietf-bess-l2vpn-yang-10 (work in progress), July 2019.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <<https://www.rfc-editor.org/info/rfc2460>>.
- [RFC4448] Martini, L., Ed., Rosen, E., El-Aawar, N., and G. Heron, "Encapsulation Methods for Transport of Ethernet over MPLS Networks", RFC 4448, DOI 10.17487/RFC4448, April 2006, <<https://www.rfc-editor.org/info/rfc4448>>.
- [RFC4761] Kompella, K., Ed. and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <<https://www.rfc-editor.org/info/rfc4761>>.
- [RFC4762] Lasserre, M., Ed. and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <<https://www.rfc-editor.org/info/rfc4762>>.

- [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>>.
- [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>>.

Appendix A. Comparison with the IEEE 802.1Q Configuration Model

In addition to the sub-interface based YANG model proposed here, the IEEE 802.1Q working group has developed a YANG model for the configuration of 802.1Q VLANs. This raises the valid question as to whether the models overlap and whether it is necessary or beneficial to have two different models for superficially similar constructs. This section aims to answer that question by summarizing and comparing the two models.

A.1. Sub-interface based configuration model overview

The key features of the sub-interface based configuration model can be summarized as:

- o The model is primarily designed to enable layer 2 and layer 3 services on Ethernet interfaces that can be defined in a very flexible way to meet the varied requirements of service providers.
- o Traffic is classified from an Ethernet-like interface to sub-interfaces based on fields in the layer 2 header. This is often based on VLAN Ids contained in the frame, but the model is extensible to other arbitrary fields in the frame header.
- o Sub-interfaces are just a type of if:interface and hence support any feature configuration YANG models that can be applied generally to interfaces. For example, QoS or ACL models that reference if:interface can be applied to the sub-interfaces, or the sub-interface can be used as an Access Circuit in L2VPN or L3VPN models that reference if:interface.

- o In the sub-interface based configuration model, the classification of traffic arriving on an interface to a given sub-interface, based on fields in the layer 2 header, is completely independent of how the traffic is forwarded. The sub-interface can be referenced (via references to if:interface) by other models that specify how traffic is forwarded; thus sub-interfaces can support multiple different forwarding paradigms, including but not limited to: layer 3 (IPv4/IPv6), layer 2 pseudowires (over MPLS or IP), VPLS instances, EVPN instance.
- o The model is flexible in the scope of the VLAN Identifier space. I.e. by default VLAN Ids can be scoped locally to a single Ethernet-like trunk interface, but the scope is determined by the forwarding paradigm that is used.

A.2. IEEE 802.1Q Bridge Configuration Model Overview

The key features of the IEEE 802.1Q bridge configuration model can be summarized as:

- o Each VLAN bridge component has a set of Ethernet interfaces that are members of that bridge. Sub-interfaces are not used, nor required in the 802.1Q bridge model.
- o Within a VLAN bridge component, the VLAN tag in the packet is used, along with the destination MAC address, to determine how to forward the packet. Other forwarding paradigms are not supported by the 802.1Q model.
- o Classification of traffic to a VLAN bridge component is based only on the Ethernet interface that it arrived on.
- o VLAN Identifiers are scoped to a VLAN bridge component. Often devices only support a single bridge component and hence VLANs are scoped globally within the device.
- o Feature configuration is specified in the context of the bridge, or particular VLANs on a bridge.

A.3. Possible Overlap Between the Two Models

Both models can be used for configuring similar basic layer 2 forwarding topologies. The 802.1Q bridge configuration model is optimised for configuring Virtual LANs that span across enterprises and data centers.

The sub-interface model can also be used for configuring equivalent Virtual LAN networks that span across enterprises and data centers,

but often requires more configuration to be able to configure the equivalent constructs to the 802.1Q bridge model.

The sub-interface model really excels when implementing flexible L2 and L3 services, where those services may be handled on the same physical interface, and where the VLAN Identifier is being solely used to identify the customer or service that is being provided rather than a Virtual LAN. The sub-interface model provides more flexibility as to how traffic can be classified, how features can be applied to traffic streams, and how the traffic is to be forwarded.

Conversely, the 802.1Q bridge model can also be use to implement L2 services in some scenarios, but only if the forwarding paradigm being used to implement the service is the native Ethernet forwarding specified in 802.1Q - other forwarding paradigms such as pseudowires or VPLS are not supported. The 802.1Q bridge model does not implement L3 services at all, although this can be partly mitigated by using a virtual L3 interface construct that is a separate logical Ethernet-like interface which is a member of the bridge.

In conclusion, it is valid for both of these models to exist since they have different deployment scenarios for which they are optimized. Devices may choose which of the models (or both) to implement depending on what functionality the device is being designed for.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@cisco.com

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2017

M. Bjorklund
Tail-f Systems
L. Berger, Ed.
LabN Consulting, L.L.C.
June 29, 2017

YANG Tree Diagrams
draft-ietf-netmod-yang-tree-diagrams-01

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

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

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

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

This Internet-Draft will expire on December 31, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Tree Diagram Syntax	3
2.1. Submodules	4
2.2. Groupings	4
2.3. Collapsed Node Representation	4
2.4. Node Representation	4
2.5. Extensions	5
3. Usage Guidelines For RFCs	5
3.1. Wrapping Long Lines	6
4. YANG Schema Mount Tree Diagrams	6
5. IANA Considerations	7
6. Informative References	7
Authors' Addresses	8

1. Introduction

YANG Tree Diagrams were first published in [RFC7223]. Such diagrams are commonly used to provide 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

```

The remainder of this document contains YANG Tree Diagram syntax based on output from pyang version 1.7.1.

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. Top level module statements are listed immediately following, offset by 4 spaces. Augmentations are listed next, offset by 2 spaces and identified by the keyword "augment" followed by the augment target node and a colon (':') character. This is followed by, RPCs which are identified by "rpcs:" and are also offset by 2 spaces. Notifications are last and are identified by "notifications:" and are also offset by 2 spaces.

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 prefaced 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 pipe ('|') character.

The full format, including spacing conventions is:

```
module: <module-name>
```

```
  +--<node>
  |   +--<node>
  |       +--<node>
  +--<node>
      +--<node>
          +--<node>
augment <target-node>:
  +--<node>
      +--<node>
      +--<node>
          +--<node>
```

```
rpcs:
  +---<node>
  +---<node>

notifications:
  +---<node>
    +---<node>
    |   +---<node>
    +---<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.

2.3. 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>
```

2.4. 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 schema mount points

<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 mounted module
- @ for a node made available via a schema mount parent reference

<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 "{...}?"

2.5. Extensions

TBD

3. Usage Guidelines For RFCs

This section provides general guidelines related to the use of tree diagrams in RFCs. This section covers [Authors' note: will cover] different types of trees and when to use them; for example, complete module trees, subtrees, trees for groupings etc.

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 '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
```

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 document is a generic mechanism that allows for mounting one data model consisting of any number of YANG modules at a specified location of another (parent) schema. Modules containing mount points will identify mount points by name using the mount-point extension. These mount-points should be identified, as indicated above using the 'mp' flag. For example:

```
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?
```

Note that a mount point definition alone is not sufficient to identify if a mount point configuration or for non-configuration data. This is determined by the yang-schema-mount module 'config' leaf associated with the specific mount point.

In describing the intended use of a module containing a mount point, it is helpful to show how the mount point would look with mounted

modules. In such cases, 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 example, the following 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/
              | ...
            +--rw rt:routing/
              | ...
            +--ro if:interfaces@
              | ...
            +--ro if:interfaces-state@
              ...

```

The with 'config' indicating 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. Informative References

- [I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-05 (work in progress), May 2017.
- [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>>.

[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>>.

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

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

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

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

This Internet-Draft will expire on August 12, 2018.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Tree Diagram Syntax	3
2.1. Submodules	6
2.2. Groupings	6
2.3. yang-data	6
2.4. Collapsed Node Representation	6
2.5. Comments	7
2.6. Node Representation	7
3. Usage Guidelines For RFCs	8
3.1. Wrapping Long Lines	8
3.2. Groupings	9
3.3. Long Diagrams	9
4. YANG Schema Mount Tree Diagrams	10
4.1. Representation of Mounted Schema Trees	10
5. IANA Considerations	12
6. Security Considerations	12
7. Informative References	12
Authors' Addresses	13

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 prefaced 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
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

L. Berger
LabN Consulting, L.L.C.
C. Hopps
Deutsche Telekom
A. Lindem
Cisco Systems
D. Bogdanovic

X. Liu
Jabil
July 3, 2017

YANG Logical Network Elements
draft-ietf-rtgwg-lne-model-03

Abstract

This document defines a logical network element module. This module can be used to manage the logical resource partitioning that may be present on a network device. Examples of common industry terms for logical resource partitioning are Logical Systems or Logical Routers.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Overview	3
3. Logical Network Elements	5
3.1. LNE Instantiation and Resource Assignment	6
3.2. LNE Management - LNE View	7
3.3. LNE Management - Host Network Device View	7
4. Security Considerations	8
5. IANA Considerations	9
6. Logical Network Element Model	9
7. References	13
7.1. Normative References	13
7.2. Informative References	13
Appendix A. Acknowledgments	14
Appendix B. Examples	15
B.1. Example: Host Device Managed LNE	15
B.1.1. Configuration Data	19
B.1.2. State Data	23
B.2. Example: Self Managed LNE	32
B.2.1. Configuration Data	35
B.2.2. State Data	38
Authors' Addresses	47

1. Introduction

This document defines a YANG [RFC6020] module to support the creation of logical network elements on a network device. A logical network element (LNE) is an independently managed virtual device made up of resources allocated to it from the host or parent network device. An LNE running on a host network device conceptually parallels a virtual machine running on a host system. Using host-virtualization terminology one could refer to an LNE as a "Guest", and the containing network-device as the "Host". While LNEs may be implemented via host-virtualization technologies this is not a requirement.

This document also defines the necessary augmentations for allocating host resources to a given LNE. As the interface management model [RFC7223] is the only a module that currently defines host resources,

this document currently defines only a single augmentation to cover the assignment of interfaces to an LNE. Future modules that define support for the control of host device resources are expected to, where appropriate, provide parallel support for the assignment of controlled resources to LNEs.

As each LNE is an independently managed device, each will have its own set of YANG modeled data that is independent of the host device and other LNEs. For example, multiple LNEs may all have their own "Tunnel0" interface defined which will not conflict with each other and will not exist in the host's interface model. An LNE will have its own management interfaces possibly including independent instances of netconf/restconf/etc servers to support configuration of their YANG models. As an example of this independence, an implementation may choose to completely rename assigned interfaces, so on the host the assigned interface might be called "Ethernet0/1" while within the LNE it might be called "eth1".

In addition to standard management interfaces, a host device implementation may support accessing LNE configuration and operational YANG models directly from the host system. When supported, such access is accomplished through a yang-schema-mount mount point [I-D.ietf-netmod-schema-mount] under which the root level LNE YANG models may be accessed.

Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

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].

Readers are expected to be familiar with terms and concepts of YANG [RFC7950] and YANG Schema Mount [I-D.ietf-netmod-schema-mount].

This document uses the graphical representation of data models defined in [I-D.ietf-netmod-yang-tree-diagrams].

2. Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g, routers, firewalls, and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical

network elements (LNEs), each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

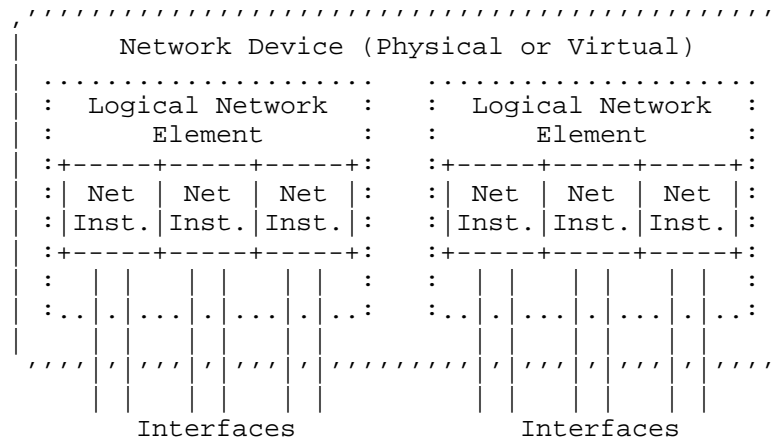


Figure 1: Module Element Relationships

A model for LNEs is described in Section 3 and the model for NIs is covered in [I-D.ietf-rtgwg-ni-model].

The interface management model [RFC7223] is an existing model that is impacted by the definition of LNEs and network instances. This document and [I-D.ietf-rtgwg-ni-model] define augmentations to the interface module to support LNEs and NIs. Similar elements, although perhaps only for LNEs, may also need to be included as part of the definition of the future hardware and QoS modules.

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [RFC7223]. The logical-network-element module augments existing interface management model by adding an identifier which is used on physical interface types to identify an associated LNE.

The interface related augmentation is as follows:

```
module: ietf-logical-network-element
  augment /if:interfaces/if:interface:
    +--rw bind-lne-name?    ->
      /logical-network-elements/logical-network-element/name
```

The interface model defined in [RFC7223] is structured to include all interfaces in a flat list, without regard to logical assignment of resources supported on the device. The bind-lne-name leaf provides the association between an interface and its associated LNE. Note that as currently defined, to assign an interface to both an LNE and NI, the interface would first be assigned to the LNE and then within that LNE's interface module, the LNE's representation of that interface would be assigned to an NI using the mechanisms defined in [I-D.ietf-rtgwg-ni-model].

3. Logical Network Elements

Logical network elements support the ability of some devices to partition resources into independent logical routers and/or switches. Device support for multiple logical network elements is implementation specific. Systems without such capabilities need not include support for the logical-network-element module. In physical devices, some hardware features are shared across partitions, but control plane (e.g., routing) protocol instances, tables, and configuration are managed separately. For example, in logical routers or VNFs, this may correspond to establishing multiple logical instances using a single software installation. The model supports configuration of multiple instances on a single device by creating a list of logical network elements, each with their own configuration and operational state related to routing and switching protocols.

The LNE model can be represented using the tree format defined in [I-D.ietf-netmod-yang-tree-diagrams] as:


```

module: ietf-logical-network-element
  +--rw logical-network-elements
    +--rw logical-network-element* [name]
      +--rw name                string
      +--rw managed?            boolean
      +--rw description?        string
      +--mp root
  augment /if:interfaces/if:interface:
    +--rw bind-lne-name?
      -> /logical-network-elements/logical-network-element/name

  notifications:
    +---n bind-lne-name-failed
      +--ro name                -> /if:interfaces/interface/name
      +--ro bind-lne-name
      |   -> /if:interfaces/interface/lne:bind-lne-name
      +--ro error-info?         string

```

'name' identifies the logical network element. 'managed' indicates if the server providing the host network device will provide the client LNE information via the 'root' structure. The root of an LNE's specific data is the schema mount point 'root'. bind-lne-name is used to associated an interface with an LNE and bind-lne-name-failed is used in certain failure cases.

An LNE root MUST contain at least the YANG library [RFC7895] and Interfaces [RFC7223] modules.

3.1. LNE Instantiation and Resource Assignment

Logical network elements may be controlled by clients using existing list operations. When list entries are created, a new LNE is instantiated. The models mounted under an LNE root are expected to be dependent on the server implementation. When a list entry is deleted, an existing LNE is destroyed. For more information, see [RFC7950] Section 7.8.6.

Once instantiated, host network device resources can be associated with the new LNE. As previously mentioned, this document augments ietf-interfaces with the bind-lne-name leaf to support such associations for interfaces. When an bind-lne-name is set to a valid LNE name, an implementation MUST take whatever steps are internally necessary to assign the interface to the LNE or provide an error message (defined below) with an indication of why the assignment failed. It is possible for the assignment to fail while processing the set, or after asynchronous processing. Error notification in the latter case is supported via a notification.

On a successful interface assignment to an LNE, an implementation MUST also make the resource available to the LNE by providing a system created interface to the LNE. The name of the system created interface is a local matter and may be identical or completely different, and mapped from and to, the name used in the context of the host device. The system created interface SHOULD be exposed via the LNE-specific instance of the interfaces module [RFC7223].

3.2. LNE Management - LNE View

Each LNE instance is expected to support management functions from within the context of the LNE root, via a server that provides information with the LNE's root exposed as device root. Management functions operating within the context of an LNE are accessed through the LNE's standard management interfaces, e.g., NETCONF and SNMP. Initial configuration, much like the initial configuration of the host device, is a local implementation matter.

When accessing an LNE via the LNE's management interface, a network-device representation will be presented, but its scope will be limited to the specific LNE. Normal YANG/NETCONF mechanisms, together with the required YANG library [RFC7895] instance, can be used to identify the available modules. Each supported module will be presented as a top level module. Only LNE associated resources will be reflected in resource related modules, e.g., interfaces, hardware, and perhaps QoS. From the management perspective, there will be no difference between the available LNE view (information) and a physical network device.

3.3. LNE Management - Host Network Device View

There are multiple implementation approaches possible to enable a network device to support the logical-network-element module and multiple LNEs. Some approaches will allow the management functions operating at network device level to access LNE configuration and operational information, while others will not. Similarly, even when LNE management from the network device is supported by the implementation, it may be prohibited by user policy.

Independent of the method selected by an implementation, the 'managed' boolean mentioned above is used to indicate when LNE management from the network device context is possible. When the 'managed' boolean is 'false', the LNE cannot be managed by the host system and can only be managed from within the context of the LNE as described in the previous section, Section 3.2. Attempts to access information below a root node whose associated 'managed' boolean is set to 'false' MUST result in the error message indicated below. In some implementations, it may not be possible to change this value.

For example, when an LNE is implemented using virtual machine and traditional hypervisor technologies, it is likely that this value will be restricted to a 'false' value.

It is an implementation choice if the information can be accessed and modified from within the context of the LNE, or even the context of the host device. When the 'managed' boolean is 'true', LNE information SHALL be accessible from the context of the host device. When the associated schema-mount definition has the 'config' leaf set to 'true', then LNE information SHALL also be modifiable from the context of the host device. When LNE information is available from both the host device and from within the context of the LNE, the same information MUST be made available via the 'root' element, with paths modified as described in [I-D.ietf-netmod-schema-mount].

An implementation MAY represent an LNE's schema using either the 'inline' or 'use-schema' approaches defined in [I-D.ietf-netmod-schema-mount]. The choice of which to use is completely an implementation choice. The inline case is anticipated to be generally used in the cases where the 'managed' will always be 'false'. The 'use-schema' approach is expected to be most useful in the case where all LNEs share the same schema. When 'use-schema' is used with an LNE mount point, the YANG library rooted in the LNE's mount point MUST match the associated schema defined within the ietf-yang-schema-mount module.

Beyond the two modules that will always be present for an LNE, as an LNE is a network device itself, all modules that may be present at the top level network device MAY also be present for the LNE. The list of available modules is expected to be implementation dependent. As is the method used by an implementation to support LNEs. Appendix B provide example uses of LNEs.

4. Security Considerations

LNE information represents device and network configuration information. As such, the security of this information is important, but it is fundamentally no different than any other interface or device configuration information that has already been covered in other documents such as [RFC7223], [RFC7317] and [RFC8022].

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

```
/logical-network-elements/logical-network-element: This list
  specifies the logical network element and the related logical
  device configuration.
```

/logical-network-elements/logical-network-element/managed: While this leaf is contained in the previous list, it is worth particular attention as it controls whether information under the LNE mount point is accessible by both the host device and within the LNE context. There may be extra sensitivity to this leaf in environments where an LNE is managed by a different party than the host device, and that party does not wish to share LNE information with the operator of the host device.

/if:interfaces/if:interface/bind-lne-name: This leaf indicates the LNE instance to which an interface is assigned.

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

5. 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-logical-network-element

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-logical-network-element
namespace: urn:ietf:params:xml:ns:yang:ietf-logical-network-element
prefix:    lne
reference:  RFC XXXX
```

6. Logical Network Element Model

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

```
<CODE BEGINS> file "ietf-logical-network-element@2017-06-30.yang"
module ietf-logical-network-element {
  yang-version 1.1;

  // namespace
```

```
namespace "urn:ietf:params:xml:ns:yang:ietf-logical-network-element";
prefix lne;

// import some basic types

import ietf-interfaces {
  prefix if;
  reference "RFC 7223: A YANG Data Model for Interface Management";
}
import ietf-yang-schema-mount {
  prefix yangmnt;
  reference "draft-ietf-netmod-schema-mount: YANG Schema Mount";
  // RFC Ed.: Please replace this draft name with the corresponding
  // RFC number
}

organization
  "IETF Routing Area (rtgwg) Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
  WG List:    <mailto:rtgwg@ietf.org>

  Author:     Lou Berger
              <mailto:lberger@labn.net>
  Author:     Christan Hopps
              <mailto:chopps@chopps.org>
  Author:     Acee Lindem
              <mailto:acee@cisco.com>
  Author:     Dean Bogdanovic
              <mailto:ivandean@gmail.com>";
description
  "This module is used to support multiple logical network
  elements on a single physical or virtual system.

  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.";

// RFC Ed.: replace XXXX with actual RFC number and remove
```

```
// this note
// RFC Ed.: please update TBD

revision 2017-03-13 {
  description
    "Initial revision.";
  reference "RFC TBD";
}

// top level device definition statements

container logical-network-elements {
  description
    "Allows a network device to support multiple logical
    network element (device) instances.";
  list logical-network-element {
    key "name";
    description
      "List of logical network elements.";
    leaf name {
      type string;
      description
        "Device-wide unique identifier for the
        logical network element.";
    }
    leaf managed {
      type boolean;
      description
        "True if the host can access LNE information
        using the root mount point. This value
        may not be modifiable in all implementations.";
    }
    leaf description {
      type string;
      description
        "Description of the logical network element.";
    }
  }
  yangmnt:mount-point "root" {
    description
      "Root for models supported per logical
      network element. This mount point will
      may or may not be inline based on the server
      implementation. It SHALL always contain a YANG
      library and interfaces instance.

      When the associated 'managed' leaf is 'false' any
      operation that attempts to access information below
      the root SHALL fail with an error-tag of
```

```
        'access-denied' and an error-app-tag of
        'lne-not-managed'.";
    }
}

// augment statements

augment "/if:interfaces/if:interface" {
    description
        "Add a node for the identification of the logical network
        element associated with an interface. Applies to interfaces
        that can be assigned on a per logical network element basis.

        Note that a standard error will be returned if the
        identified leafref isn't present. If an interfaces cannot
        be assigned for any other reason, the operation SHALL fail
        with an error-tag of 'operation-failed' and an error-app-tag
        of 'lne-assignment-failed'. A meaningful error-info that
        indicates the source of the assignment failure SHOULD also
        be provided.";
    leaf bind-lne-name {
        type leafref {
            path "/logical-network-elements/logical-network-element/name";
        }
        description
            "Logical network element ID to which interface is bound.";
    }
}

// notification statements

notification bind-lne-name-failed {
    description
        "Indicates an error in the association of an interface to an
        LNE. Only generated after success is initially returned when
        bind-lne-name is set.";
    leaf name {
        type leafref {
            path "/if:interfaces/if:interface/if:name";
        }
        mandatory true;
        description
            "Contains the interface name associated with the
            failure.";
    }
    leaf bind-lne-name {
        type leafref {
```

```
    path "/if:interfaces/if:interface/lne:bind-lne-name";
  }
  mandatory true;
  description
    "Contains the bind-lne-name associated with the
    failure.";
}
leaf error-info {
  type string;
  description
    "Optionally, indicates the source of the assignment
    failure.";
}
}
}
<CODE ENDS>
```

7. References

7.1. Normative References

- [I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-05 (work in progress), May 2017.
- [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>>.
- [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>>.

7.2. 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.
- [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-ni-model]
Berger, L., Hopps, C., Lindem, A., and D. Bogdanovic, "YANG Network Instances", draft-ietf-rtgwg-ni-model-02 (work in progress), March 2017.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.
- [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>>.
- [RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <<http://www.rfc-editor.org/info/rfc8022>>.

Appendix A. Acknowledgments

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Yan Gang. Useful review comments were also received by Martin Bjorklund and John Scudder.

This document was motivated by, and derived from, [I-D.ietf-rtgwg-device-model].

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

Appendix B. Examples

The following subsections provide example uses of LNEs.

B.1. Example: Host Device Managed LNE

This section describes an example of the LNE model using schema mount to achieve the parent management. An example device supports multiple instances of LNEs (logical routers), each of which supports features of layer 2 and layer 3 interfaces [RFC7223], routing information base [RFC8022], and OSPF protocol. Each of these features is specified by a YANG model, and they are combined using YANG Schema Mount as follows:

```
module: ietf-logical-network-element
  +--rw logical-network-elements
    +--rw logical-network-element* [name]
      +--rw name string
      +--mp root
        +--ro yanglib:modules-state/
          | +--ro module-set-id string
          | +--ro module* [name revision]
          |   +--ro name yang:yang-identifier
        +--rw sys:system/
          | +--rw contact? string
          | +--rw hostname? inet:domain-name
          | +--rw authentication {authentication}?
          |   +--rw user-authentication-order* identityref
          |   +--rw user* [name] {local-users}?
          |     +--rw name string
          |     +--rw password? ianach:crypt-hash
          |     +--rw authorized-key* [name]
          |       +--rw name string
          |       +--rw algorithm string
          |       +--rw key-data binary
          +--ro sys:system-state/
          |   ...
          +--ro rt:routing-state/
          |   +--ro router-id? yang:dotted-quad
          |   +--ro control-plane-protocols
          |     +--ro control-plane-protocol* [type name]
          |       +--ro ospf:ospf/
          |         +--ro instance* [af]
          |         ...
          +--rw rt:routing/
          |   +--rw router-id? yang:dotted-quad
          |   +--rw control-plane-protocols
          |     +--rw control-plane-protocol* [type name]
```

```

|         +---rw ospf:ospf/
|             +---rw instance* [af]
|                 +---rw areas
|                     +---rw area* [area-id]
|                         +---rw interfaces
|                             +---rw interface* [name]
|                                 +---rw name if:interface-ref
|                                 +---rw cost?      uint16
+---rw if:interfaces/
|   +---rw interface* [name]
|       +---rw name                string
|       +---rw ip:ipv4!/
|           | +---rw address* [ip]
|           | ...
+---ro if:interfaces-state/
    +---ro interface* [name]
        +---ro name                string
        +---ro ip:ipv4!/
            | +---ro address* [ip]
            | ...

module: ietf-interfaces
+---rw interfaces
|   +---rw interface* [name]
|       +---rw name                string
|       +---rw lne:bind-lne-name?  string
+---ro interfaces-state
    +---ro interface* [name]
        +---ro name                string
        +---ro oper-status          enumeration

module: ietf-yang-library
+---ro modules-state
    +---ro module-set-id            string
    +---ro module* [name revision]
        +---ro name                  yang:yang-identifier

module: ietf-system
+---rw system
|   +---rw contact?                string
|   +---rw hostname?               inet:domain-name
|   +---rw authentication {authentication}?
|       +---rw user-authentication-order* identityref
|       +---rw user* [name] {local-users}?
|           +---rw name              string
|           +---rw password?          ianach:crypt-hash
|           +---rw authorized-key* [name]
|               +---rw name            string

```

```

|           +--rw algorithm    string
|           +--rw key-data     binary
+--ro system-state
  +--ro platform
    |   +--ro os-name?         string
    |   +--ro os-release?     string

```

To realize the above schema, the example device implements the following schema mount instance:

```

"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-logical-network-element",
      "name": "root",
      "use-schema": [
        {
          "name": "lne-schema"
        }
      ]
    }
  ],
  "schema": [
    {
      "name": "lne-schema",
      "module": [
        {
          "name": "ietf-yang-library",
          "revision": "2016-06-21",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-yang-library",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-system",
          "revision": "2014-08-06",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-system",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-routing",
          "revision": "2016-11-04",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-routing",
          "conformance-type": "implement"
        }
      ]
    }
  ]
}

```

```

        "name": "ietf-ospf",
        "revision": "2017-03-12",
        "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-ospf",
        "conformance-type": "implement"
    },
    {
        "name": "ietf-interfaces",
        "revision": "2014-05-08",
        "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"
    }
]
}
]
}

```

By using the implementation of the YANG schema mount, an operator can create instances of logical routers. An interface can be assigned to a logical router, so that the logical router has the permission to access this interface. The OSPF protocol can then be enabled on this assigned interface.

For this implementation, a parent management session has access to the schemas of both the parent hosting system and the child logical routers. In addition, each child logical router can grant its own management sessions, which have the following schema:

```

module: ietf-yang-library
  +--ro modules-state
    +--ro module-set-id      string
    +--ro module* [name revision]
      +--ro name                yang:yang-identifier

module: ietf-system
  +--rw system
    | +--rw contact?           string
    | +--rw hostname?         inet:domain-name
    | +--rw authentication {authentication}?
    |   +--rw user-authentication-order*  identityref

```

```

    |         +---rw user* [name] {local-users}?
    |         +---rw name                string
    |         +---rw password?           ianach:crypt-hash
    |         +---rw authorized-key* [name]
    |         +---rw name                string
    |         +---rw algorithm           string
    |         +---rw key-data            binary
+---ro system-state
  +---ro platform
  |   +---ro os-name?    string
  |   +---ro os-release? string
module: ietf-routing
+---ro routing-state
|   +---ro router-id?                yang:dotted-quad
|   +---ro control-plane-protocols
|   |   +---ro control-plane-protocol* [type name]
|   |   +---ro ospf:ospf/
|   |   +---ro instance* [af]
+---rw routing
  +---rw router-id?                yang:dotted-quad
  +---rw control-plane-protocols
  |   +---rw control-plane-protocol* [type name]
  |   +---rw ospf:ospf/
  |   +---rw instance* [af]
  |   +---rw areas
  |   |   +---rw area* [area-id]
  |   |   +---rw interfaces
  |   |   |   +---rw interface* [name]
  |   |   |   +---rw name        if:interface-ref
  |   |   |   +---rw cost?      uint16
module: ietf-interfaces
+---rw interfaces
|   +---rw interface* [name]
|   |   +---rw name                string
+---ro interfaces-state
  +---ro interface* [name]
  |   +---ro name                string
  |   +---ro oper-status         enumeration

```

B.1.1. Configuration Data

The following shows an example where two customer specific LNEs are configured:

```

{
  "ietf-logical-network-element:logical-network-elements": {

```

```

"logical-network-element": [
  {
    "name": "cust1",
    "root": {
      "ietf-system:system": {
        "authentication": {
          "user": [
            {
              "name": "john",
              "password": "$0$password"
            }
          ]
        }
      }
    },
    "ietf-routing:routing": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:ospf",
            "name": "1",
            "ietf-ospf:ospf": {
              "instance": [
                {
                  "af": "ipv4",
                  "areas": {
                    "area": [
                      {
                        "area-id": "203.0.113.1",
                        "interfaces": {
                          "interface": [
                            {
                              "name": "eth1",
                              "cost": 10
                            }
                          ]
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
    },
    "ietf-interfaces:interfaces": {

```

```

    "interfaces": {
      "interface": [
        {
          "name": "eth1",
          "ip:ipv4": {
            "address": [
              {
                "ip": "192.0.2.11",
                "prefix-length": 24,
              }
            ]
          }
        }
      ]
    }
  },
  {
    "name": "cust2",
    "root": {
      "ietf-system:system": {
        "authentication": {
          "user": [
            {
              "name": "john",
              "password": "$0$password"
            }
          ]
        }
      }
    }
  },
  "ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "instance": [
              {
                "af": "ipv4",
                "areas": {
                  "area": [
                    {
                      "area-id": "203.0.113.1",
                      "interfaces": {
                        "interface": [

```



```
{
    "name": "eth1",
    "cost": 10
}
]
}
]
}
]
}
]
}
]
}
]
}
]
}
]
}
]
}
]
}
],
},
{
    "ip:ipv4": {
        "address": [
            {
                "ip": "192.0.2.11",
                "prefix-length": 24,
            }
        ]
    },
    "interface": [
        {
            "name": "eth0",
            "ip:ipv4": {
                "address": [
                    {
                        "ip": "192.0.2.10",
                        "prefix-length": 24,
                    }
                ]
            }
        }
    ],
}
```

```

    }
  },
  {
    "name": "cust1:eth1",
    "lne:bind-lne-name": "cust1"
  },
  {
    "name": "cust2:eth1",
    "lne:bind-lne-name": "cust2"
  }
]
}
},
"ietf-system:system": {
  "authentication": {
    "user": [
      {
        "name": "root",
        "password": "$0$password"
      }
    ]
  }
}
}
}

```

B.1.1.2. State Data

The following shows possible state data associated the above configuration data:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "cust1",
        "root": {
          "ietf-yang-library:modules-state": {
            "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
            "module": [
              {
                "name": "iana-if-type",
                "revision": "2014-05-08",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:iana-if-type",
                "conformance-type": "import"
              },
              {

```

```
"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-ospf",
"revision": "2017-03-12",
"namespace":
"urn:ietf:params:xml:ns:yang:ietf-ospf",
"conformance-type": "implement"
},
{
"name": "ietf-routing",
"revision": "2016-11-04",
"namespace":
"urn:ietf:params:xml:ns:yang:ietf-routing",
"conformance-type": "implement"
},
{
"name": "ietf-system",
"revision": "2014-08-06",
"namespace":
"urn:ietf:params:xml:ns:yang:ietf-system",
"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-types",
        "revision": "2013-07-15",
        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
        "conformance-type": "import"
    }
]
}
"ietf-system:system-state": {
    "ietf-system:system-state": {
        "platform": {
            "os-name": "NetworkOS"
        }
    }
},
"ietf-routing:routing-state": {
    "router-id": "192.0.2.1",
    "control-plane-protocols": {
        "control-plane-protocol": [
            {
                "type": "ietf-routing:ospf",
                "name": "1",
                "ietf-ospf:ospf": {
                    "instance": [
                        {
                            "af": "ipv4",
                            "areas": {
                                "area": [
                                    {
                                        "area-id": "203.0.113.1",
                                        "interfaces": {
                                            "interface": [
                                                {
                                                    "name": "eth1",
                                                    "cost": 10
                                                }
                                            ]
                                        }
                                    }
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    }
}
]

```

```

    }
  }
]
},
"ietf-interfaces:interfaces-state": {
  "interfaces": {
    "interface": [
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
}
},
{
  "name": "cust2",
  "root": {
    "ietf-yang-library:modules-state": {
      "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
      "module": [
        {
          "name": "iana-if-type",
          "revision": "2014-05-08",
          "namespace":
            "urn:ietf:params:xml:ns:yang:iana-if-type",
          "conformance-type": "import"
        },
        {
          "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-ospf",
    "revision": "2017-03-12",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-ospf",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-routing",
    "revision": "2016-11-04",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-system",
    "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-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
      "conformance-type": "import"
    }
  ]
}
"ietf-system:system-state": {
  "ietf-system:system-state": {
    "platform": {
      "os-name": "NetworkOS"
    }
  }
},
"ietf-routing:routing-state": {
  "router-id": "192.0.2.2",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "instance": [
            {
              "af": "ipv4",
              "areas": {
                "area": [
                  {
                    "area-id": "203.0.113.1",
                    "interfaces": {
                      "interface": [
                        {
                          "name": "eth1",
                          "cost": 10
                        }
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}

```

```

    }
    "ietf-interfaces:interfaces-state": {
      "interfaces": {
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "oper-status": "up",
          "phys-address": "00:01:02:A1:B1:C2",
          "statistics": {
            "discontinuity-time": "2017-06-26T12:34:56-05:00"
          },
          "ip:ipv4": {
            "address": [
              {
                "ip": "192.0.2.11",
                "prefix-length": 24,
              }
            ]
          }
        }
      ]
    }
  }
},
"ietf-interfaces:interfaces-state": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C0",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
},
{

```



```
    "name": "cust1:eth1",
    "type": "iana-if-type:ethernetCsmacd",
    "oper-status": "up",
    "phys-address": "00:01:02:A1:B1:C1",
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    }
  },
  {
    "name": "cust2:eth1",
    "type": "iana-if-type:ethernetCsmacd",
    "oper-status": "up",
    "phys-address": "00:01:02:A1:B1:C2",
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    }
  }
]
}
},
"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "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": "2017-03-13",
      "feature": [
        "bind-lne-name"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ospf",
      "revision": "2017-03-12",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ospf",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-routing",
      "revision": "2016-11-04",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-system",
      "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"
  }
]
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
}
}
```

B.2. Example: Self Managed LNE

This section describes an example of the LNE model using schema mount to achieve child independent management. An example device supports multiple instances of LNEs (logical routers), each of them has the features of layer 2 and layer 3 interfaces [RFC7223], routing information base [RFC8022], and the OSPF protocol. Each of these features is specified by a YANG model, and they are put together by the YANG Schema Mount as following:

```

    module: ietf-logical-network-element
    +--rw logical-network-elements
      +--rw logical-network-element* [name]
        +--rw name          string
        +--mp root
          // The internal modules of the LNE are not visible to
          // the parent management.
          // The child manages its modules, including ietf-routing
          // and ietf-interfaces

module: ietf-interfaces
  +--rw interfaces
    |   +--rw interface* [name]
    |     +--rw name          string
    |     +--rw lne:bind-lne-name?  string
  +--ro interfaces-state
    +--ro interface* [name]
      +--ro name          string
      +--ro oper-status   enumeration

module: ietf-yang-library
  +--ro modules-state
    +--ro module-set-id   string
    +--ro module* [name revision]
      +--ro name          yang:yang-identifier

module: ietf-system
  +--rw system
    |   +--rw contact?          string
    |   +--rw hostname?         inet:domain-name
    |   +--rw authentication {authentication}?
    |     +--rw user-authentication-order*  identityref
    |     +--rw user* [name] {local-users}?
    |       +--rw name          string
    |       +--rw password?     ianach:crypt-hash
    |       +--rw authorized-key* [name]
    |         +--rw name        string
    |         +--rw algorithm   string
    |         +--rw key-data    binary
  +--ro system-state
    +--ro platform
      |   +--ro os-name?       string
      |   +--ro os-release?    string

```

To realize the above schema, the device implements the following schema mount instance:

```

"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-logical-network-element",
      "name": "root",
      "inline": [null]
    }
  ]
}

```

By using the implementation of the YANG schema mount, an operator can create instances of logical routers, each with their logical router specific in-line modules. An interface can be assigned to a logical router, so that the logical router has the permission to access this interface. The OSPF protocol can then be enabled on this assigned interface. Each logical router independently manages its own set of modules, which may or may not be the same as other logical routers. The following is an example of schema set implemented on one particular logical router:

```

module: ietf-yang-library
  +--ro modules-state
    +--ro module-set-id    string
    +--ro module* [name revision]
      +--ro name                yang:yang-identifier

module: ietf-system
  +--rw system
    | +--rw contact?          string
    | +--rw hostname?        inet:domain-name
    | +--rw authentication {authentication}?
    | | +--rw user-authentication-order* identityref
    | | +--rw user* [name] {local-users}?
    | | | +--rw name          string
    | | | +--rw password?     ianach:crypt-hash
    | | | +--rw authorized-key* [name]
    | | | | +--rw name        string
    | | | | +--rw algorithm   string
    | | | | +--rw key-data    binary
    | +--ro system-state
    | | +--ro platform
    | | | +--ro os-name?      string
    | | | +--ro os-release?   string

module: ietf-routing
  +--ro routing-state
    | +--ro router-id?                yang:dotted-quad
    | +--ro control-plane-protocols

```

```

| | | +--ro control-plane-protocol* [type name]
| | |   +--ro ospf:ospf/
| | |     +--ro instance* [af]
+--rw routing
  +--rw router-id?                yang:dotted-quad
  +--rw control-plane-protocols
    +--rw control-plane-protocol* [type name]
    +--rw ospf:ospf/
      +--rw instance* [af]
      +--rw areas
        +--rw area* [area-id]
        +--rw interfaces
          +--rw interface* [name]
            +--rw name          if:interface-ref
            +--rw cost?        uint16

module: ietf-interfaces
+--rw interfaces
| +--rw interface* [name]
| +--rw name                string
+--ro interfaces-state
  +--ro interface* [name]
  +--ro name                string
  +--ro oper-status          enumeration

```

B.2.1. Configuration Data

Each of the child virtual routers is managed through its own sessions and configuration data.

B.2.1.1. Logical Network Element 'vnf1'

The following shows an example configuration data for the LNE name "vnf1":

```

{
  "ietf-system:system": {
    "authentication": {
      "user": [
        {
          "name": "john",
          "password": "$0$password"
        }
      ]
    }
  },
  "ietf-routing:routing": {
    "router-id": "192.0.2.1",

```

```

"control-plane-protocols": {
  "control-plane-protocol": [
    {
      "type": "ietf-routing:ospf",
      "name": "1",
      "ietf-ospf:ospf": {
        "instance": [
          {
            "af": "ipv4",
            "areas": {
              "area": [
                {
                  "area-id": "203.0.113.1",
                  "interfaces": {
                    "interface": [
                      {
                        "name": "eth1",
                        "cost": 10
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
    }
  ],
  "ietf-interfaces:interfaces": {
    "interfaces": {
      "interface": [
        {
          "name": "eth1",
          "ip:ipv4": {
            "address": [
              {
                "ip": "192.0.2.11",
                "prefix-length": 24,
              }
            ]
          }
        }
      ]
    }
  }
}

```

```
}
```

B.2.1.2. Logical Network Element 'vnf2'

The following shows an example configuration data for the LNE name "vnf2":

```
{
  "ietf-system:system": {
    "authentication": {
      "user": [
        {
          "name": "john",
          "password": "$0$password"
        }
      ]
    }
  },
  "ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "instance": [
              {
                "af": "ipv4",
                "areas": {
                  "area": [
                    {
                      "area-id": "203.0.113.1",
                      "interfaces": {
                        "interface": [
                          {
                            "name": "eth1",
                            "cost": 10
                          }
                        ]
                      }
                    }
                  ]
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```



```

    ]
  },
  "ietf-interfaces:interfaces": {
    "interfaces": {
      "interface": [
        {
          "name": "eth1",
          "ip:ipv4": {
            "address": [
              {
                "ip": "192.0.2.11",
                "prefix-length": 24,
              }
            ]
          }
        }
      ]
    }
  }
}

```

B.2.2. State Data

The following sections shows possible state data associated the above configuration data. Note that there are three views: the host device's, and each LNE's.

B.2.2.1. Host Device

The following shows state data for the device hosting the example LNEs:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "vnf1",
        "root": {
        }
      },
      {
        "name": "vnf2",
        "root": {
        }
      }
    ]
  }
},

```

```
"ietf-interfaces:interfaces-state": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C0",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        }
      },
      {
        "name": "vnf1:eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        }
      },
      {
        "name": "vnf2:eth2",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C2",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        }
      }
    ]
  }
},

"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
    }
  ]
}
```

```
    "namespace":  
    "urn:ietf:params:xml:ns:yang:iana-if-type",  
    "conformance-type": "import"  
  },  
  {  
    "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": "2017-03-13",  
    "feature": [  
      "bind-lne-name"  
    ],  
    "namespace":  
    "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",  
    "conformance-type": "implement"  
  },  
  {  
    "name": "ietf-system",  
    "revision": "2014-08-06",  
    "namespace":  
    "urn:ietf:params:xml:ns:yang:ietf-system",  
    "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"
    }
]
},
"ietf-system:system-state": {
    "platform": {
        "os-name": "NetworkOS"
    }
}
}

```

B.2.2.2. Logical Network Element 'vnf1'

The following shows state data for the example LNE with name "vnf1":

```

{
    "ietf-yang-library:modules-state": {
        "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
        "module": [
            {
                "name": "iana-if-type",
                "revision": "2014-05-08",
                "namespace":
                "urn:ietf:params:xml:ns:yang:iana-if-type",
                "conformance-type": "import"
            },
            {
                "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-ospf",
    "revision": "2017-03-12",
    "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-ospf",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-routing",
    "revision": "2016-11-04",
    "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-system",
    "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-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
      "conformance-type": "import"
    }
  ]
},

"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},

"ietf-routing:routing-state": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "instance": [
            {
              "af": "ipv4",
              "areas": {
                "area": [
                  {
                    "area-id": "203.0.113.1",
                    "interfaces": {
                      "interface": [
                        {
                          "name": "eth1",
                          "cost": 10
                        }
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}
]

```

```

    }
  },
  "ietf-interfaces:interfaces-state": {
    "interfaces": {
      "interface": [
        {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "oper-status": "up",
          "phys-address": "00:01:02:A1:B1:C1",
          "statistics": {
            "discontinuity-time": "2017-06-26T12:34:56-05:00"
          },
          "ip:ipv4": {
            "address": [
              {
                "ip": "192.0.2.11",
                "prefix-length": 24,
              }
            ]
          }
        }
      ]
    }
  }
}

```

B.2.2.3. Logical Network Element 'vnf2'

The following shows state data for the example LNE with name "vnf2":

```

{
  "ietf-yang-library:modules-state": {
    "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace":
          "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "import"
      },
      {
        "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-ospf",
    "revision": "2017-03-12",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-ospf",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-routing",
    "revision": "2016-11-04",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-system",
    "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-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
      "conformance-type": "import"
    }
  ]
},

"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},

"ietf-routing:routing-state": {
  "router-id": "192.0.2.2",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "instance": [
            {
              "af": "ipv4",
              "areas": {
                "area": [
                  {
                    "area-id": "203.0.113.1",
                    "interfaces": {
                      "interface": [
                        {
                          "name": "eth1",
                          "cost": 10
                        }
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}

```

```
    },  
    "ietf-interfaces:interfaces-state": {  
      "interfaces": {  
        "interface": [  
          {  
            "name": "eth1",  
            "type": "iana-if-type:ethernetCsmacd",  
            "oper-status": "up",  
            "phys-address": "00:01:02:A1:B1:C2",  
            "statistics": {  
              "discontinuity-time": "2017-06-26T12:34:56-05:00"  
            },  
            "ip:ipv4": {  
              "address": [  
                {  
                  "ip": "192.0.2.11",  
                  "prefix-length": 24,  
                }  
              ]  
            }  
          }  
        ]  
      }  
    }  
  }  
}
```

Authors' Addresses

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

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

Email: acee@cisco.com

Dean Bogdanovic

Email: ivandean@gmail.com

Xufeng Liu

Jabil

Email: Xufeng_Liu@jabil.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2018

L. Berger
LabN Consulting, L.L.C.
C. Hopps
Deutsche Telekom
A. Lindem
Cisco Systems
D. Bogdanovic

X. Liu
Jabil
March 19, 2018

YANG Model for Logical Network Elements
draft-ietf-rtgwg-lne-model-10

Abstract

This document defines a logical network element YANG module. This module can be used to manage the logical resource partitioning that may be present on a network device. Examples of common industry terms for logical resource partitioning are Logical Systems or Logical Routers. The YANG model in this document conforms to the Network Management Datastore Architecture as defined in RFCXXXX.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 20, 2018.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Overview	4
3. Logical Network Elements	5
3.1. LNE Instantiation and Resource Assignment	6
3.2. LNE Management - LNE View	7
3.3. LNE Management - Host Network Device View	7
4. Security Considerations	8
5. IANA Considerations	9
6. Logical Network Element Model	10
7. References	14
7.1. Normative References	14
7.2. Informative References	15
Appendix A. Acknowledgments	16
Appendix B. Examples	17
B.1. Example: Host Device Managed LNE	17
B.1.1. Configuration Data	20
B.1.2. State Data	24
B.2. Example: Self Managed LNE	33
B.2.1. Configuration Data	36
B.2.2. State Data	39
Authors' Addresses	48

1. Introduction

This document defines a YANG [RFC6020] module to support the creation of logical network elements on a network device. A logical network element (LNE) is an independently managed virtual device made up of resources allocated to it from the host or parent network device. An LNE running on a host network device conceptually parallels a virtual machine running on a host system. Using host-virtualization terminology one could refer to an LNE as a "Guest", and the containing network-device as the "Host". While LNEs may be implemented via host-virtualization technologies this is not a requirement. The YANG model in this document conforms to the Network

Management Datastore Architecture defined in the [I-D.ietf-netmod-revised-datastores].

This document also defines the necessary augmentations for allocating host resources to a given LNE. As the interface management model [I-D.ietf-netmod-rfc7223bis] is the only a module that currently defines host resources, this document currently defines only a single augmentation to cover the assignment of interfaces to an LNE. Future modules that define support for the control of host device resources are expected to, where appropriate, provide parallel support for the assignment of controlled resources to LNEs.

As each LNE is an independently managed device, each will have its own set of YANG modeled data that is independent of the host device and other LNEs. For example, multiple LNEs may all have their own "Tunnel0" interface defined which will not conflict with each other and will not exist in the host's interface model. An LNE will have its own management interfaces possibly including independent instances of netconf/restconf/etc servers to support configuration of their YANG models. As an example of this independence, an implementation may choose to completely rename assigned interfaces, so on the host the assigned interface might be called "Ethernet0/1" while within the LNE it might be called "eth1".

In addition to standard management interfaces, a host device implementation may support accessing LNE configuration and operational YANG models directly from the host system. When supported, such access is accomplished through a yang-schema-mount mount point [I-D.ietf-netmod-schema-mount] under which the root level LNE YANG models may be accessed.

Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

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] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with terms and concepts of YANG [RFC7950] and YANG Schema Mount [I-D.ietf-netmod-schema-mount].

This document uses the graphical representation of data models defined in [I-D.ietf-netmod-yang-tree-diagrams].

2. Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g., routers, firewalls, and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs), each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

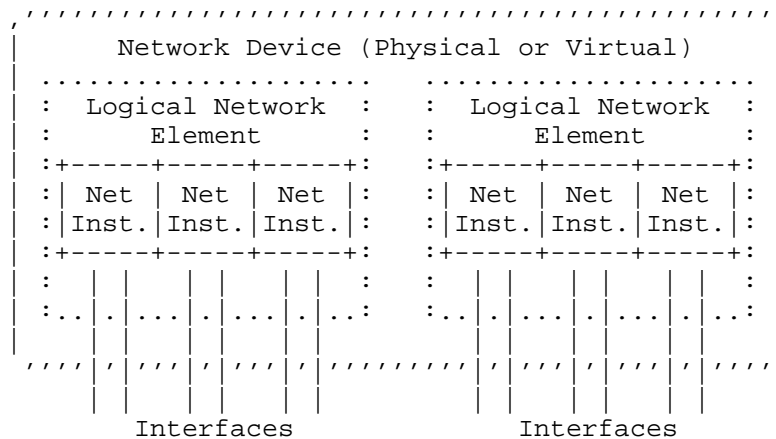


Figure 1: Module Element Relationships

A model for LNEs is described in Section 3 and the model for NIs is covered in [I-D.ietf-rtgwg-ni-model].

The interface management model [I-D.ietf-netmod-rfc7223bis] is an existing model that is impacted by the definition of LNEs and network instances. This document and [I-D.ietf-rtgwg-ni-model] define augmentations to the interface module to support LNEs and NIs. Similar elements, although perhaps only for LNEs, may also need to be included as part of the definition of the future hardware and QoS modules.

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw

physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [I-D.ietf-netmod-rfc7223bis]. The logical-network-element module augments existing interface management model by adding an identifier which is used on interfaces to identify an associated LNE.

The interface related augmentation is as follows:

```
module: ietf-logical-network-element
  augment /if:interfaces/if:interface:
    +---rw bind-lne-name?    ->
      /logical-network-elements/logical-network-element/name
```

The interface model defined in [I-D.ietf-netmod-rfc7223bis] is structured to include all interfaces in a flat list, without regard to logical assignment of resources supported on the device. The bind-lne-name leaf provides the association between an interface and its associated LNE. Note that as currently defined, to assign an interface to both an LNE and NI, the interface would first be assigned to the LNE and then within that LNE's interface module, the LNE's representation of that interface would be assigned to an NI using the mechanisms defined in [I-D.ietf-rtgwg-ni-model].

3. Logical Network Elements

Logical network elements support the ability of some devices to partition resources into independent logical routers and/or switches. Device support for multiple logical network elements is implementation specific. Systems without such capabilities need not include support for the logical-network-element module. In physical devices, some hardware features are shared across partitions, but control plane (e.g., routing) protocol instances, tables, and configuration are managed separately. For example, in logical routers or VNFs, this may correspond to establishing multiple logical instances using a single software installation. The model supports configuration of multiple instances on a single device by creating a list of logical network elements, each with their own configuration and operational state related to routing and switching protocols.

The LNE model can be represented as:


```
module: ietf-logical-network-element
  +--rw logical-network-elements
    +--rw logical-network-element* [name]
      +--rw name                string
      +--rw managed?           boolean
      +--rw description?       string
      +--mp root
  augment /if:interfaces/if:interface:
    +--rw bind-lne-name?
      -> /logical-network-elements/logical-network-element/name

  notifications:
    +---n bind-lne-name-failed
      +--ro name                -> /if:interfaces/interface/name
      +--ro bind-lne-name
      |   -> /if:interfaces/interface/lne:bind-lne-name
      +--ro error-info?         string
```

'name' identifies the logical network element. 'managed' indicates if the server providing the host network device will provide the client LNE information via the 'root' structure. The root of an LNE's specific data is the schema mount point 'root'. bind-lne-name is used to associated an interface with an LNE and bind-lne-name-failed is used in certain failure cases.

An LNE root MUST contain at least the YANG library [RFC7895] and Interfaces [I-D.ietf-netmod-rfc7223bis] modules.

3.1. LNE Instantiation and Resource Assignment

Logical network elements may be controlled by clients using existing list operations. When list entries are created, a new LNE is instantiated. The models mounted under an LNE root are expected to be dependent on the server implementation. When a list entry is deleted, an existing LNE is destroyed. For more information, see [RFC7950] Section 7.8.6.

Once instantiated, host network device resources can be associated with the new LNE. As previously mentioned, this document augments ietf-interfaces with the bind-lne-name leaf to support such associations for interfaces. When an bind-lne-name is set to a valid LNE name, an implementation MUST take whatever steps are internally necessary to assign the interface to the LNE or provide an error message (defined below) with an indication of why the assignment failed. It is possible for the assignment to fail while processing the set, or after asynchronous processing. Error notification in the latter case is supported via a notification.

On a successful interface assignment to an LNE, an implementation MUST also make the resource available to the LNE by providing a system created interface to the LNE. The name of the system created interface is a local matter and may be identical or completely different, and mapped from and to, the name used in the context of the host device. The system created interface SHOULD be exposed via the LNE-specific instance of the interfaces module [I-D.ietf-netmod-rfc7223bis].

3.2. LNE Management - LNE View

Each LNE instance is expected to support management functions from within the context of the LNE root, via a server that provides information with the LNE's root exposed as device root. Management functions operating within the context of an LNE are accessed through the LNE's standard management interfaces, e.g., NETCONF and SNMP. Initial configuration, much like the initial configuration of the host device, is a local implementation matter.

When accessing an LNE via the LNE's management interface, a network-device representation will be presented, but its scope will be limited to the specific LNE. Normal YANG/NETCONF mechanisms, together with the required YANG library [RFC7895] instance, can be used to identify the available modules. Each supported module will be presented as a top level module. Only LNE associated resources will be reflected in resource related modules, e.g., interfaces, hardware, and perhaps QoS. From the management perspective, there will be no difference between the available LNE view (information) and a physical network device.

3.3. LNE Management - Host Network Device View

There are multiple implementation approaches possible to enable a network device to support the logical-network-element module and multiple LNEs. Some approaches will allow the management functions operating at network device level to access LNE configuration and operational information, while others will not. Similarly, even when LNE management from the network device is supported by the implementation, it may be prohibited by user policy.

Independent of the method selected by an implementation, the 'managed' boolean mentioned above is used to indicate when LNE management from the network device context is possible. When the 'managed' boolean is 'false', the LNE cannot be managed by the host system and can only be managed from within the context of the LNE as described in the previous section, Section 3.2. Attempts to access information below a root node whose associated 'managed' boolean is set to 'false' MUST result in the error message indicated below. In

some implementations, it may not be possible to change this value. For example, when an LNE is implemented using virtual machine and traditional hypervisor technologies, it is likely that this value will be restricted to a 'false' value.

It is an implementation choice if the information can be accessed and modified from within the context of the LNE, or even the context of the host device. When the 'managed' boolean is 'true', LNE information SHALL be accessible from the context of the host device. When the associated schema-mount definition has the 'config' leaf set to 'true', then LNE information SHALL also be modifiable from the context of the host device. When LNE information is available from both the host device and from within the context of the LNE, the same information MUST be made available via the 'root' element, with paths modified as described in [I-D.ietf-netmod-schema-mount].

An implementation MAY represent an LNE's schema using either the 'inline' or 'shared-schema' approaches defined in [I-D.ietf-netmod-schema-mount]. The choice of which to use is completely an implementation choice. The inline case is anticipated to be generally used in the cases where the 'managed' will always be 'false'. The 'shared-schema' approach is expected to be most useful in the case where all LNEs share the same schema. When 'shared-schema' is used with an LNE mount point, the YANG library rooted in the LNE's mount point MUST match the associated schema defined according to the ietf-yang-schema-mount module.

Beyond the two modules that will always be present for an LNE, as an LNE is a network device itself, all modules that may be present at the top level network device MAY also be present for the LNE. The list of available modules is expected to be implementation dependent. As is the method used by an implementation to support LNEs. Appendix B provide example uses of LNEs.

4. 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.

LNE information represents device and network configuration information. As such, the security of this information is important, but it is fundamentally no different than any other interface or device configuration information that has already been covered in other documents such as [I-D.ietf-netmod-rfc7223bis], [RFC7317] and [I-D.ietf-netmod-rfc8022bis].

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

/logical-network-elements/logical-network-element: This list specifies the logical network element and the related logical device configuration.

/logical-network-elements/logical-network-element/managed: While this leaf is contained in the previous list, it is worth particular attention as it controls whether information under the LNE mount point is accessible by both the host device and within the LNE context. There may be extra sensitivity to this leaf in environments where an LNE is managed by a different party than the host device, and that party does not wish to share LNE information with the operator of the host device.

/if:interfaces/if:interface/bind-lne-name: This leaf indicates the LNE instance to which an interface is assigned. Implementations should pay particular attention to when changes to this leaf are permitted as removal of an interface from an LNE can have major impact on the LNEs operation as it is similar to physically removing an interface from the device. Implementations can reject an reassignment using the previously described error message generation.

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

5. 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-logical-network-element

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-logical-network-element
namespace:    urn:ietf:params:xml:ns:yang:ietf-logical-network-element
prefix:       lne
reference:     RFC XXXX
```

6. Logical Network Element Model

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

```
<CODE BEGINS> file "ietf-logical-network-element@2018-03-20.yang"
module ietf-logical-network-element {
  yang-version 1.1;

  // namespace

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-logical-network-element";
  prefix lne;

  // import some basic types

  import ietf-interfaces {
    prefix if;
    reference "draft-ietf-netmod-rfc7223bis:
              A YANG Data Model for Interface Management";
  }
  import ietf-yang-schema-mount {
    prefix yangmnt;
    reference "draft-ietf-netmod-schema-mount: YANG Schema Mount";
    // RFC Ed.: Please replace this draft name with the corresponding
    // RFC number
  }

  organization
    "IETF Routing Area (rtgwg) Working Group";
  contact
    "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
    WG List:    <mailto:rtgwg@ietf.org>

    Author:     Lou Berger
                <mailto:lberger@labn.net>
    Author:     Christan Hopps
                <mailto:chopps@chopps.org>
    Author:     Acee Lindem
```

```

    Author:      <mailto:acee@cisco.com>
                Dean Bogdanovic
                <mailto:ivandean@gmail.com>";
description
  "This module is used to support multiple logical network
  elements on a single physical or virtual system.

  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.";

// RFC Ed.: replace XXXX with actual RFC number and remove
// this note
// RFC Ed.: please update TBD

revision 2018-03-20 {
  description
    "Initial revision.";
  reference "RFC XXXX";
}

// top level device definition statements

container logical-network-elements {
  description
    "Allows a network device to support multiple logical
    network element (device) instances.";
  list logical-network-element {
    key "name";
    description
      "List of logical network elements.";
    leaf name {
      type string;
      description
        "Device-wide unique identifier for the
        logical network element.";
    }
    leaf managed {
      type boolean;
    }
  }
}
```

```
    default "true";
    description
        "True if the host can access LNE information
         using the root mount point. This value
         my not be modifiable in all implementations.";
}
leaf description {
    type string;
    description
        "Description of the logical network element.";
}
container "root" {
    description
        "Container for mount point.";
    yangmnt:mount-point "root" {
        description
            "Root for models supported per logical
             network element. This mount point may or may not
             be inline based on the server implementation. It
             SHALL always contain a YANG library and interfaces
             instance.

             When the associated 'managed' leaf is 'false' any
             operation that attempts to access information below
             the root SHALL fail with an error-tag of
             'access-denied' and an error-app-tag of
             'lne-not-managed'.";
    }
}
}
}

// augment statements

augment "/if:interfaces/if:interface" {
    description
        "Add a node for the identification of the logical network
         element associated with an interface. Applies to
         interfaces that can be assigned on a per logical network
         element basis.

        Note that a standard error will be returned if the
        identified leafref isn't present. If an interfaces
        cannot be assigned for any other reason, the operation
        SHALL fail with an error-tag of 'operation-failed' and an
        error-app-tag of 'lne-assignment-failed'. A meaningful
        error-info that indicates the source of the assignment
        failure SHOULD also be provided.";
```

```
    leaf bind-lne-name {
      type leafref {
        path
          "/logical-network-elements/logical-network-element/name";
      }
      description
        "Logical network element ID to which interface is bound.";
    }
  }

  // notification statements

  notification bind-lne-name-failed {
    description
      "Indicates an error in the association of an interface to an
       LNE. Only generated after success is initially returned when
       bind-lne-name is set.";
    leaf name {
      type leafref {
        path "/if:interfaces/if:interface/if:name";
      }
      mandatory true;
      description
        "Contains the interface name associated with the
         failure.";
    }
    leaf bind-lne-name {
      type leafref {
        path "/if:interfaces/if:interface/lne:bind-lne-name";
      }
      mandatory true;
      description
        "Contains the bind-lne-name associated with the
         failure.";
    }
    leaf error-info {
      type string;
      description
        "Optionally, indicates the source of the assignment
         failure.";
    }
  }
}
<CODE ENDS>
```


7. References

7.1. 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-10
(work in progress), January 2018.
- [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-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-
ietf-netmod-schema-mount-08 (work in progress), October
2017.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
DOI 10.17487/RFC3688, January 2004,
<<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol Version 1.2", RFC 5246,
DOI 10.17487/RFC5246, August 2008,
<<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for
the Network Configuration Protocol (NETCONF)", RFC 6020,
DOI 10.17487/RFC6020, October 2010,
<<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed.,
and A. Bierman, Ed., "Network Configuration Protocol
(NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,
<<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure
Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011,
<<https://www.rfc-editor.org/info/rfc6242>>.

- [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>>.
- [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>>.

7.2. Informative References

- [I-D.ietf-netmod-entity]
Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", draft-ietf-netmod-entity-08 (work in progress), January 2018.
- [I-D.ietf-netmod-rfc8022bis]
Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", draft-ietf-netmod-rfc8022bis-11 (work in progress), January 2018.
- [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.
- [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-ni-model]
Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Model for Network Instances", draft-ietf-rtgwg-ni-model-11 (work in progress), March 2018.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [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>>.

Appendix A. Acknowledgments

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Yan Gang. Useful review comments were also received by Martin Bjorklund, John Scudder, Dan Romascanu and Taylor Yu.

This document was motivated by, and derived from, [I-D.ietf-rtgwg-device-model].

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

Thanks to Alvaro Retana for IESG review.

Appendix B. Examples

The following subsections provide example uses of LNEs.

B.1. Example: Host Device Managed LNE

This section describes an example of the LNE model using schema mount to achieve the parent management. An example device supports multiple instances of LNEs (logical routers), each of which supports features of layer 2 and layer 3 interfaces [I-D.ietf-netmod-rfc7223bis], routing information base [I-D.ietf-netmod-rfc8022bis], and OSPF protocol. Each of these features is specified by a YANG model, and they are combined using YANG Schema Mount as shown below. Not all possible mounted modules are shown. For example implementations could also mount the model defined in [I-D.ietf-netmod-entity].

```

module: ietf-logical-network-element
  +--rw logical-network-elements
    +--rw logical-network-element* [name]
      +--rw name string
      +--mp root
        +--ro yanglib:modules-state/
          |   +--ro module-set-id string
          |   +--ro module* [name revision]
          |   |   +--ro name yang:yang-identifier
          +--rw sys:system/
            |   +--rw contact? string
            |   +--rw hostname? inet:domain-name
            |   +--rw authentication {authentication}?
            |   |   +--rw user-authentication-order* identityref
            |   |   +--rw user* [name] {local-users}?
            |   |   |   +--rw name string
            |   |   |   +--rw password? ianach:crypt-hash
            |   |   |   +--rw authorized-key* [name]
            |   |   |   |   +--rw name string
            |   |   |   |   +--rw algorithm string
            |   |   |   |   +--rw key-data binary
            +--ro sys:system-state/
              |   ...
            +--rw rt:routing/
              |   +--rw router-id? yang:dotted-quad
              |   +--rw control-plane-protocols
              |   |   +--rw control-plane-protocol* [type name]
              |   |   |   +--rw ospf:ospf/
              |   |   |   |   +--rw areas

```

```

|           |--rw area* [area-id]
|           |--rw interfaces
|             |--rw interface* [name]
|               |--rw name if:interface-ref
|               |--rw cost?   uint16
|--rw if:interfaces/
  |--rw interface* [name]
    |--rw name          string
    |--rw ip:ipv4!/
    |   |--rw address* [ip]
    |   ...
module: ietf-interfaces
  |--rw interfaces
    |--rw interface* [name]
      |--rw name          string
      |--rw lne:bind-lne-name?  string
      |--ro oper-status    enumeration
module: ietf-yang-library
  |--ro modules-state
    |--ro module-set-id    string
    |--ro module* [name revision]
      |--ro name          yang:yang-identifier
module: ietf-system
  |--rw system
    |--rw contact?        string
    |--rw hostname?       inet:domain-name
    |--rw authentication {authentication}?
      |--rw user-authentication-order*  identityref
      |--rw user* [name] {local-users}?
        |--rw name          string
        |--rw password?     ianach:crypt-hash
        |--rw authorized-key* [name]
          |--rw name          string
          |--rw algorithm    string
          |--rw key-data     binary
  |--ro system-state
    |--ro platform
      |--ro os-name?        string
      |--ro os-release?     string

```

To realize the above schema, the example device implements the following schema mount instance:

```
"ietf-yang-schema-mount:schema-mounts": {  
  "mount-point": [  
    {  
      "module": "ietf-logical-network-element",  
      "label": "root",  
      "shared-schema": {}  
    }  
  ]  
}
```

By using the implementation of the YANG schema mount, an operator can create instances of logical routers. An interface can be assigned to a logical router, so that the logical router has the permission to access this interface. The OSPF protocol can then be enabled on this assigned interface.

For this implementation, a parent management session has access to the schemas of both the parent hosting system and the child logical routers. In addition, each child logical router can grant its own management sessions, which have the following schema:

```

module: ietf-yang-library
  +--ro modules-state
    +--ro module-set-id      string
    +--ro module* [name revision]
      +--ro name                yang:yang-identifier

module: ietf-system
  +--rw system
    | +--rw contact?          string
    | +--rw hostname?        inet:domain-name
    | +--rw authentication {authentication}?
    | | +--rw user-authentication-order* identityref
    | | +--rw user* [name] {local-users}?
    | | | +--rw name          string
    | | | +--rw password?     ianach:crypt-hash
    | | | +--rw authorized-key* [name]
    | | | | +--rw name        string
    | | | | +--rw algorithm   string
    | | | | +--rw key-data    binary
    | +--ro system-state
    | +--ro platform
    | | +--ro os-name?       string
    | | +--ro os-release?    string

module: ietf-routing
  rw-- routing
    +--rw router-id?          yang:dotted-quad
    +--rw control-plane-protocols
    | +--rw control-plane-protocol* [type name]
    | | +--rw ospf:ospf/
    | | | +--rw areas
    | | | | +--rw area* [area-id]
    | | | | +--rw interfaces
    | | | | | +--rw interface* [name]
    | | | | | | +--rw name      if:interface-ref
    | | | | | | +--rw cost?    uint16

module: ietf-interfaces
  +--rw interfaces
    +--rw interface* [name]
    | +--rw name                string
    +--ro oper-status           enumeration

```

B.1.1. Configuration Data

The following shows an example where two customer specific LNEs are configured:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "cust1",
        "root": {
          "ietf-system:system": {
            "authentication": {
              "user": [
                {
                  "name": "john",
                  "password": "$0$password"
                }
              ]
            }
          },
          "ietf-routing:routing": {
            "router-id": "192.0.2.1",
            "control-plane-protocols": {
              "control-plane-protocol": [
                {
                  "type": "ietf-routing:ospf",
                  "name": "1",
                  "ietf-ospf:ospf": {
                    "af": "ipv4",
                    "areas": {
                      "area": [
                        {
                          "area-id": "203.0.113.1",
                          "interfaces": {
                            "interface": [
                              {
                                "name": "eth1",
                                "cost": 10
                              }
                            ]
                          }
                        }
                      ]
                    }
                  }
                }
              ]
            }
          },
          "ietf-interfaces:interfaces": {
            "interfaces": {
              "interface": [

```



```

    {
      "name": "eth1",
      "ip:ipv4": {
        "address": [
          {
            "ip": "192.0.2.11",
            "prefix-length": 24,
          }
        ]
      },
      "ip:ipv6": {
        "address": [
          {
            "ip": "2001:db8:0:2::11",
            "prefix-length": 64,
          }
        ]
      }
    }
  ]
}
},
{
  "name": "cust2",
  "root": {
    "ietf-system:system": {
      "authentication": {
        "user": [
          {
            "name": "john",
            "password": "$0$password"
          }
        ]
      }
    }
  }
  "ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "af": "ipv4",
            "areas": {
              "area": [

```

```
{
    "area-id": "203.0.113.1",
    "interfaces": {
        "interface": [
            {
                "name": "eth1",
                "cost": 10
            }
        ]
    }
}

],
{
    "ietf-interfaces:interfaces": {
        "interfaces": {
            {
                "name": "eth1",
                "ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.11",
                            "prefix-length": 24,
                        }
                    ]
                }
            }
        ]
    }
},
{
    "ietf-interfaces:interfaces": {
        "interfaces": {
            "interface": [
                {
                    "name": "eth0",
                    "ip:ipv4": {
                        "address": [
                            {
                                "ip": "192.0.2.10",
                                "prefix-length": 24,
```

```

    }
  ]
},
"ip:ipv6": {
  "address": [
    {
      "ip": "2001:db8:0:2::10",
      "prefix-length": 64,
    }
  ]
}
},
{
  "name": "cust1:eth1",
  "lne:bind-lne-name": "cust1"
},
{
  "name": "cust2:eth1",
  "lne:bind-lne-name": "cust2"
}
]
}
},
"ietf-system:system": {
  "authentication": {
    "user": [
      {
        "name": "root",
        "password": "$0$password"
      }
    ]
  }
}
}
}

```

B.1.2. State Data

The following shows possible state data associated the above configuration data:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "cust1",
        "root": {
          "ietf-yang-library:modules-state": {

```

```
"module-set-id": "123e4567-e89b-12d3-a456-426655440000",
"module": [
  {
    "name": "iana-if-type",
    "revision": "2014-05-08",
    "namespace":
      "urn:ietf:params:xml:ns:yang:iana-if-type",
    "conformance-type": "import"
  },
  {
    "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-ospf",
    "revision": "2018-03-03",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-ospf",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-routing",
    "revision": "2018-03-13",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
]
```

```
{
  "name": "ietf-system",
  "revision": "2014-08-06",
  "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-system",
  "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-types",
  "revision": "2013-07-15",
  "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-yang-types",
  "conformance-type": "import"
}
]
},
"ietf-system:system-state": {
  "ietf-system:system-state": {
    "platform": {
      "os-name": "NetworkOS"
    }
  }
},
"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
                "area-id": "203.0.113.1",
                "interfaces": {
                  "interface": [
                    {
                      "name": "eth1",
                      "cost": 10
                    }
                  ]
                }
              }
            ]
          }
        }
      }
    ]
  }
}
```

```

    }
  ]
}
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
},
{
  "name": "cust2",
  "root": {
    "ietf-yang-library:modules-state": {
      "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
      "module": [
        {
          "name": "iana-if-type",
          "revision": "2014-05-08",
          "namespace":
            "urn:ietf:params:xml:ns:yang:iana-if-type",
          "conformance-type": "import"
        }
      ]
    }
  }
}

```

```
    },
    {
      "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-ospf",
      "revision": "2018-03-03",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ospf",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-routing",
      "revision": "2018-03-13",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-system",
      "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-types",
        "revision": "2013-07-15",
        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
        "conformance-type": "import"
    }
]
}
"ietf-system:system-state": {
    "ietf-system:system-state": {
        "platform": {
            "os-name": "NetworkOS"
        }
    }
},
"ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
        "control-plane-protocol": [
            {
                "type": "ietf-routing:ospf",
                "name": "1",
                "ietf-ospf:ospf": {
                    "af": "ipv4",
                    "areas": {
                        "area": [
                            {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                    "interface": [
                                        {
                                            "name": "eth1",
                                            "cost": 10
                                        }
                                    ]
                                }
                            }
                        ]
                    }
                }
            }
        ]
    }
}

```



```

    ]
  }
}
"ietf-interfaces:interfaces": {
  "interfaces": {
    {
      "name": "eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "oper-status": "up",
      "phys-address": "00:01:02:A1:B1:C2",
      "statistics": {
        "discontinuity-time": "2017-06-26T12:34:56-05:00"
      },
      "ip:ipv4": {
        "address": [
          {
            "ip": "192.0.2.11",
            "prefix-length": 24,
          }
        ]
      }
    }
  ]
}
}
]
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C0",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
}

```

```
    },
    {
      "name": "cust1:eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "oper-status": "up",
      "phys-address": "00:01:02:A1:B1:C1",
      "statistics": {
        "discontinuity-time": "2017-06-26T12:34:56-05:00"
      }
    },
    {
      "name": "cust2:eth1",
      "type": "iana-if-type:ethernetCsmacd",
      "oper-status": "up",
      "phys-address": "00:01:02:A1:B1:C2",
      "statistics": {
        "discontinuity-time": "2017-06-26T12:34:56-05:00"
      }
    }
  ]
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},
"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "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": "2017-03-13",
  "feature": [
    "bind-lne-name"
  ],
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
  "conformance-type": "implement"
},
{
  "name": "ietf-ospf",
  "revision": "2018-03-03",
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-ospf",
  "conformance-type": "implement"
},
{
  "name": "ietf-routing",
  "revision": "2018-03-13",
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-routing",
  "conformance-type": "implement"
},
{
  "name": "ietf-system",
  "revision": "2014-08-06",
  "namespace":
"urn:ietf:params:xml:ns:yang:ietf-system",
  "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"
    }
]
},
"ietf-yang-schema-mount:schema-mounts": {
    "mount-point": [
        {
            "module": "ietf-logical-network-element",
            "label": "root",
            "shared-schema": {}
        }
    ]
}
}

```

B.2. Example: Self Managed LNE

This section describes an example of the LNE model using schema mount to achieve child independent management. An example device supports multiple instances of LNEs (logical routers), each of them has the features of layer 2 and layer 3 interfaces [I-D.ietf-netmod-rfc7223bis], routing information base [I-D.ietf-netmod-rfc8022bis], and the OSPF protocol. Each of these features is specified by a YANG model, and they are put together by the YANG Schema Mount as following:

```

    module: ietf-logical-network-element
+--rw logical-network-elements
  +--rw logical-network-element* [name]
    +--rw name                string
    +--mp root
      // The internal modules of the LNE are not visible to
      // the parament management.
      // The child manages its modules, including ietf-routing
      // and ietf-interfaces

module: ietf-interfaces
+--rw interfaces
  +--rw interface* [name]
    +--rw name                string
    +--rw lne:bind-lne-name?   string
    +--ro oper-status          enumeration

module: ietf-yang-library
+--ro modules-state
  +--ro module-set-id         string
  +--ro module* [name revision]
    +--ro name                yang:yang-identifier

module: ietf-system
+--rw system
|   +--rw contact?            string
|   +--rw hostname?          inet:domain-name
|   +--rw authentication {authentication}?
|     +--rw user-authentication-order* identityref
|     +--rw user* [name] {local-users}?
|       +--rw name            string
|       +--rw password?       ianach:crypt-hash
|       +--rw authorized-key* [name]
|         +--rw name          string
|         +--rw algorithm     string
|         +--rw key-data      binary
+--ro system-state
  +--ro platform
    |   +--ro os-name?        string
    |   +--ro os-release?     string

```

To realize the above schema, the device implements the following schema mount instance:

```
"ietf-yang-schema-mount:schema-mounts": {  
  "mount-point": [  
    {  
      "module": "ietf-logical-network-element",  
      "label": "root",  
      "inline": {}  
    }  
  ]  
}
```

By using the implementation of the YANG schema mount, an operator can create instances of logical routers, each with their logical router specific in-line modules. An interface can be assigned to a logical router, so that the logical router has the permission to access this interface. The OSPF protocol can then be enabled on this assigned interface. Each logical router independently manages its own set of modules, which may or may not be the same as other logical routers. The following is an example of schema set implemented on one particular logical router:

```

module: ietf-yang-library
  +--ro modules-state
    +--ro module-set-id      string
    +--ro module* [name revision]
      +--ro name                yang:yang-identifier

module: ietf-system
  +--rw system
    | +--rw contact?          string
    | +--rw hostname?         inet:domain-name
    | +--rw authentication {authentication}?
    |   +--rw user-authentication-order* identityref
    |   +--rw user* [name] {local-users}?
    |     +--rw name          string
    |     +--rw password?     ianach:crypt-hash
    |     +--rw authorized-key* [name]
    |       +--rw name        string
    |       +--rw algorithm   string
    |       +--rw key-data    binary
    +--ro system-state
      +--ro platform
        | +--ro os-name?      string
        | +--ro os-release?   string

module: ietf-routing
  +--rw routing
    +--rw router-id?          yang:dotted-quad
    +--rw control-plane-protocols
      +--rw control-plane-protocol* [type name]
        +--rw ospf:ospf/
          +--rw areas
            +--rw area* [area-id]
              +--rw interfaces
                +--rw interface* [name]
                  +--rw name      if:interface-ref
                  +--rw cost?     uint16

module: ietf-interfaces
  +--rw interfaces
    +--rw interface* [name]
      +--rw name                string
      +--ro oper-status         enumeration

```

B.2.1. Configuration Data

Each of the child virtual routers is managed through its own sessions and configuration data.

B.2.1.1.1. Logical Network Element 'vnf1'

The following shows an example configuration data for the LNE name "vnf1":

```
{
  "ietf-system:system": {
    "authentication": {
      "user": [
        {
          "name": "john",
          "password": "$0$password"
        }
      ]
    }
  },
  "ietf-routing:routing": {
    "router-id": "192.0.2.1",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "af": "ipv4",
            "areas": {
              "area": [
                {
                  "area-id": "203.0.113.1",
                  "interfaces": {
                    "interface": [
                      {
                        "name": "eth1",
                        "cost": 10
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
      ]
    }
  },
  "ietf-interfaces:interfaces": {
    "interfaces": {
      "interface": [
```



```

    {
      "name": "eth1",
      "ip:ipv4": {
        "address": [
          {
            "ip": "192.0.2.11",
            "prefix-length": 24,
          }
        ]
      }
    }
  ]
}

```

B.2.1.2. Logical Network Element 'vnf2'

The following shows an example configuration data for the LNE name "vnf2":

```

{
  "ietf-system:system": {
    "authentication": {
      "user": [
        {
          "name": "john",
          "password": "$0$password"
        }
      ]
    }
  },
  "ietf-routing:routing": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
      "control-plane-protocol": [
        {
          "type": "ietf-routing:ospf",
          "name": "1",
          "ietf-ospf:ospf": {
            "af": "ipv4",
            "areas": {
              "area": [
                {
                  "area-id": "203.0.113.1",
                  "interfaces": {
                    "interface": [
                      {

```

```

        "name": "eth1",
        "cost": 10
      }
    ]
  }
}
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth1",
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
}
}

```

B.2.2. State Data

The following sections shows possible state data associated the above configuration data. Note that there are three views: the host device's, and each LNE's.

B.2.2.1. Host Device

The following shows state data for the device hosting the example LNEs:

```

{
  "ietf-logical-network-element:logical-network-elements": {
    "logical-network-element": [
      {
        "name": "vnf1",

```

```
        "root": {
        }
    },
    {
        "name": "vnf2",
        "root": {
        }
    }
]
},
"ietf-interfaces:interfaces": {
    "interfaces": {
        "interface": [
            {
                "name": "eth0",
                "type": "iana-if-type:ethernetCsmacd",
                "oper-status": "up",
                "phys-address": "00:01:02:A1:B1:C0",
                "statistics": {
                    "discontinuity-time": "2017-06-26T12:34:56-05:00"
                },
                "ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.10",
                            "prefix-length": 24,
                        }
                    ]
                }
            }
        ],
        {
            "name": "vnf1:eth1",
            "type": "iana-if-type:ethernetCsmacd",
            "oper-status": "up",
            "phys-address": "00:01:02:A1:B1:C1",
            "statistics": {
                "discontinuity-time": "2017-06-26T12:34:56-05:00"
            }
        },
        {
            "name": "vnf2:eth2",
            "type": "iana-if-type:ethernetCsmacd",
            "oper-status": "up",
            "phys-address": "00:01:02:A1:B1:C2",
            "statistics": {
                "discontinuity-time": "2017-06-26T12:34:56-05:00"
            }
        }
    ]
}
```

```
    }
  ]
}
},

"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},

"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "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": "2017-03-13",
    "feature": [
      "bind-lne-name"
    ],
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-logical-network-element",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-system",
    "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"
  }
]
},
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-logical-network-element",
      "label": "root",
      "inline": {}
    }
  ]
}
```

```
}
```

B.2.2.2. Logical Network Element 'vnf1'

The following shows state data for the example LNE with name "vnf1":

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace":
          "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "import"
      },
      {
        "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-ospf",
        "revision": "2018-03-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ospf",
        "conformance-type": "implement"
      }
    ]
  }
}
```

```
    },
    {
      "name": "ietf-routing",
      "revision": "2018-03-13",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-system",
      "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-types",
      "revision": "2013-07-15",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-yang-types",
      "conformance-type": "import"
    }
  ]
},

"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},

"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
```

```

        "area": [
          {
            "area-id": "203.0.113.1",
            "interfaces": {
              "interface": [
                {
                  "name": "eth1",
                  "cost": 10
                }
              ]
            }
          }
        ]
      }
    }
  }
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      }
    ]
  }
}
}

```


B.2.2.3. Logical Network Element 'vnf2'

The following shows state data for the example LNE with name "vnf2":

```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
    "module": [
      {
        "name": "iana-if-type",
        "revision": "2014-05-08",
        "namespace":
          "urn:ietf:params:xml:ns:yang:iana-if-type",
        "conformance-type": "import"
      },
      {
        "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-ospf",
        "revision": "2018-03-03",
        "namespace":
          "urn:ietf:params:xml:ns:yang:ietf-ospf",
        "conformance-type": "implement"
      }
    ]
  }
}
```

```
    "name": "ietf-routing",
    "revision": "2018-03-13",
    "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-system",
    "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-types",
    "revision": "2013-07-15",
    "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-yang-types",
    "conformance-type": "import"
  }
]
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
},
"ietf-routing:routing": {
  "router-id": "192.0.2.2",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
```

```

        "area-id": "203.0.113.1",
        "interfaces": {
            "interface": [
                {
                    "name": "eth1",
                    "cost": 10
                }
            ]
        }
    }
}

},

"ietf-interfaces:interfaces": {
    "interfaces": {
        "interface": [
            {
                "name": "eth1",
                "type": "iana-if-type:ethernetCsmacd",
                "oper-status": "up",
                "phys-address": "00:01:02:A1:B1:C2",
                "statistics": {
                    "discontinuity-time": "2017-06-26T12:34:56-05:00"
                },
                "ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.0.2.11",
                            "prefix-length": 24,
                        }
                    ]
                }
            }
        ]
    }
}
}

```

Authors' Addresses

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

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

Email: acee@cisco.com

Dean Bogdanovic

Email: ivandean@gmail.com

Xufeng Liu
Jabil

Email: Xufeng_Liu@jabil.com

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

L. Berger
LabN Consulting, L.L.C.
C. Hopps
Deutsche Telekom
A. Lindem
Cisco Systems
D. Bogdanovic

X. Liu
Jabil
July 3, 2017

YANG Network Instances
draft-ietf-rtgwg-ni-model-03

Abstract

This document defines a network instance module. This module can be used to manage the virtual resource partitioning that may be present on a network device. Examples of common industry terms for virtual resource partitioning are Virtual Routing and Forwarding (VRF) instances and Virtual Switch Instances (VSIs).

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
1.2. Status of Work and Open Issues	3
2. Overview	4
3. Network Instances	5
3.1. NI Types and Mount Points	6
3.1.1. Well Known Mount Points	7
3.1.2. NI Type Example	8
3.2. NIs and Interfaces	9
3.3. Network Instance Management	11
3.4. Network Instance Instantiation	13
4. Security Considerations	13
5. IANA Considerations	14
6. Network Instance Model	14
7. References	20
7.1. Normative References	20
7.2. Informative References	21
Appendix A. Acknowledgments	22
Appendix B. Example NI usage	22
B.1. Configuration Data	22
B.2. State Data	25
Authors' Addresses	30

1. Introduction

This document defines the second of two new modules that are defined to support the configuration and operation of network-devices that allow for the partitioning of resources from both, or either, management and networking perspectives. Both leverage the YANG functionality enabled by YANG Schema Mount [I-D.ietf-netmod-schema-mount].

The first form of resource partitioning provides a logical partitioning of a network device where each partition is separately managed as essentially an independent network element which is 'hosted' by the base network device. These hosted network elements are referred to as logical network elements, or LNEs, and are supported by the logical-network-element module defined in

[I-D.ietf-rtgwg-lne-model]. That module is used to identify LNEs and associate resources from the network-device with each LNE. LNEs themselves are represented in YANG as independent network devices; each accessed independently. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

The second form, which is defined in this document, provides support for what is commonly referred to as Virtual Routing and Forwarding (VRF) instances as well as Virtual Switch Instances (VSI), see [RFC4026] and [RFC4664]. In this form of resource partitioning, multiple control plane and forwarding/bridging instances are provided by and managed via a single (physical or logical) network device. This form of resource partitioning is referred to as a Network Instance and is supported by the network-instance module defined below. Configuration and operation of each network-instance is always via the network device and the network-instance module.

One notable difference between the LNE model and the NI model is that the NI model provides a framework for VRF and VSI management. This document envisions the separate definition of VRF and VSI, i.e., L3 and L2 VPN, technology specific models. An example of such can be found in the emerging L3VPN model defined in [I-D.ietf-bess-l3vpn-yang] and the examples discussed below.

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].

Readers are expected to be familiar with terms and concepts of YANG [RFC7950] and YANG Schema Mount [I-D.ietf-netmod-schema-mount].

This document uses the graphical representation of data models defined in [I-D.ietf-netmod-yang-tree-diagrams].

1.2. Status of Work and Open Issues

The top open issues are:

1. Schema mount currently doesn't allow parent-reference filtering on the instance of the mount point, but rather just the schema. This means it is not possible to filter based on actual data, e.g., `bind-network-instance-name="green"`. In the schema mount definition, the text and examples should be updated to cover this case.

2. Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g., routers, firewalls, and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs) each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

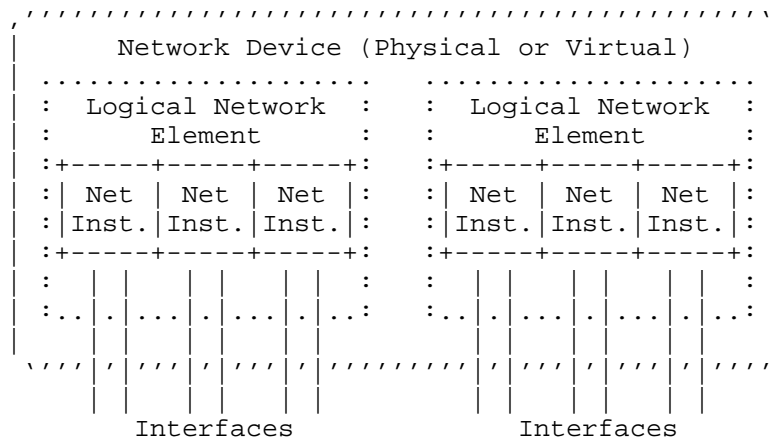


Figure 1: Module Element Relationships

A model for LNEs is described in [I-D.ietf-rtgwg-lne-model] and the model for NIs is covered in this document in Section 3.

The current interface management model [RFC7223] is impacted by the definition of LNEs and NIs. This document and [I-D.ietf-rtgwg-lne-model] define augmentations to the interface module to support LNEs and NIs.

The network instance model supports the configuration of VRFs and VSIs. Each instance is supported by information that relates to the device, for example the route target used when advertising VRF routes via the mechanisms defined in [RFC4364], and information that relates to the internal operation of the NI, for example for routing

protocols [RFC8022] and OSPF [I-D.ietf-ospf-yang]. This document defines the network-instance module that provides a basis for the management of both types of information.

NI information that relates to the device, including the assignment of interfaces to NIs, is defined as part of this document. The defined module also provides a placeholder for the definition of NI-technology specific information both at the device level and for NI internal operation. Information related to NI internal operation is supported via schema mount [I-D.ietf-netmod-schema-mount] and mounting appropriate modules under the mount point. Well known mount points are defined for L3VPN, L2VPN, and L2+L3VPN NI types.

3. Network Instances

The network instance container is used to represent virtual routing and forwarding instances (VRFs) and virtual switching instances (VSIs). VRFs and VSIs are commonly used to isolate routing and switching domains, for example to create virtual private networks, each with their own active protocols and routing/switching policies. The model supports both core/provider and virtual instances. Core/provider instance information is accessible at the top level of the server, while virtual instance information is accessible under the root schema mount points.

The NI model can be represented using the tree format defined in [I-D.ietf-netmod-yang-tree-diagrams] as:

```

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?
        +--:(vsi-root)
          | +--mp vsi-root?
        +--:(vv-root)
          +--mp vv-root?
  augment /if:interfaces/if:interface:
    +--rw bind-ni-name? -> /network-instances/network-instance/name
  augment /if:interfaces/if:interface/ip:ipv4:
    +--rw bind-ni-name? -> /network-instances/network-instance/name
  augment /if:interfaces/if:interface/ip:ipv6:
    +--rw bind-ni-name? -> /network-instances/network-instance/name

notifications:
  +---n bind-ni-name-failed
    +--ro name                -> /if:interfaces/interface/name
    +--ro interface
      | +--ro bind-ni-name?
      |                               -> /if:interfaces/interface/ni:bind-ni-name
    +--ro ipv4
      | +--ro bind-ni-name?
      |                               -> /if:interfaces/interface/ip:ipv4/ni:bind-ni-name
    +--ro ipv6
      | +--ro bind-ni-name?
      |                               -> /if:interfaces/interface/ip:ipv6/ni:bind-ni-name
    +--ro error-info?         string

```

A network instance is identified by a 'name' string. This string is used both as an index within the network-instance module and to associate resources with a network instance as shown above in the interface augmentation. The ni-type and root-type choice statements are used to support different types of L2 and L3 VPN technologies. The bind-ni-name-failed notification is used in certain failure cases.

3.1. NI Types and Mount Points

The network-instance module is structured to facilitate the definition of information models for specific types of VRFs and VSIs using augmentations. For example, the information needed to support

VPLS, VxLAN and EVPN based L2VPNs are likely to be quite different. Example models under development that could be restructured to take advantage on NIs include, for L3VPNs [I-D.ietf-bess-l3vpn-yang] and for L2VPNs [I-D.ietf-bess-l2vpn-yang].

Documents defining new YANG models for the support of specific types of network instances should augment the network instance module. The basic structure that should be used for such augmentations include a case statement, with containers for configuration and state data and finally, when needed, a type specific mount point. Generally ni types, are expected to not need to define type specific mount points, but rather reuse one of the well known mount point, as defined in the next section. The following is an example type specific augmentation:

```
augment "/ni:network-instances/ni:network-instance/ni:ni-type" {
  case l3vpn {
    container l3vpn {
      ...
    }
    container l3vpn-state {
      ...
    }
  }
}
```

3.1.1.1. Well Known Mount Points

YANG Schema Mount, [I-D.ietf-netmod-schema-mount], identifies mount points by name within a module. This definition allows for the definition of mount points whose schema can be shared across ni-types. As discussed above, ni-types largely differ in the configuration information needed in the core/top level instance to support the NI, rather than in the information represented within an NI. This allows the use of shared mount points across certain NI types.

The expectation is that there are actually very few different schema that need to be defined to support NIs on an implementation. In particular, it is expected that the following three forms of NI schema are needed, and each can be defined with a well known mount point that can be reused by future modules defining ni-types.

The three well known mount points are:

vrf-root
vrf-root is intended for use with L3VPN type ni-types.

vsi-root

vsi-root is intended for use with L2VPN type ni-types.

vv-root

vv-root is intended for use with ni-types that simultaneously support L2VPN bridging and L3VPN routing capabilities.

Future model definitions should use the above mount points whenever possible. When a well known mount point isn't appropriate, a model may define a type specific mount point via augmentation.

3.1.2. NI Type Example

The following is an example of an L3VPN VRF using a hypothetical augmentation to the networking instance schema defined in [I-D.ietf-bess-l3vpn-yang]. More detailed examples can be found in Appendix B.

```

module: ietf-network-instance
  +--rw network-instances
    +--rw network-instance* [name]
      +--rw name                string
      +--rw enabled?            boolean
      +--rw description?        string
      +--rw (ni-type)?
        |   +--:(l3vpn)
        |   |   +--rw l3vpn:l3vpn
        |   |   |   ... // config data
        |   |   +--ro l3vpn:l3vpn-state
        |   |   |   ... // state data
        |   +--rw (root-type)?
        |   |   +--:(vrf-root)
        |   |   +--mp vrf-root
        |   |   |   +--ro rt:routing-state/
        |   |   |   |   +--ro router-id?                yang:dotted-quad
        |   |   |   |   +--ro control-plane-protocols
        |   |   |   |   |   +--ro control-plane-protocol* [type name]
        |   |   |   |   |   |   +--ro ospf:ospf/
        |   |   |   |   |   |   |   +--ro instance* [af]
        |   |   |   +--rw rt:routing/
        |   |   |   |   +--rw router-id?                yang:dotted-quad
        |   |   |   |   +--rw control-plane-protocols
        |   |   |   |   |   +--rw control-plane-protocol* [type name]
        |   |   |   |   |   +--rw ospf:ospf/
        |   |   |   |   |   |   +--rw instance* [af]
        |   |   |   |   |   |   +--rw areas
        |   |   |   |   |   |   |   +--rw area* [area-id]
        |   |   |   |   |   |   |   |   +--rw interfaces
        |   |   |   |   |   |   |   |   |   +--rw interface* [name]
        |   |   |   |   |   |   |   |   |   |   +--rw name if:interface-ref
        |   |   |   |   |   |   |   |   |   |   +--rw cost?    uint16
        |   |   |   +--ro if:interfaces@
        |   |   |   |   ...
        |   |   |   +--ro if:interfaces-state@
        |   |   |   |   ...

```

This shows YANG Routing Management [RFC8022] and YANG OSPF [I-D.ietf-ospf-yang] as mounted modules. The mounted modules can reference interface information via a parent-reference to the containers defined in [RFC7223].

3.2. NIs and Interfaces

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration,

and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [RFC7223].

As shown below, the network-instance module augments the existing interface management model by adding a name which is used on interface or sub-interface types to identify an associated network instance. Similarly, this name is also added for IPv4 and IPv6 types, as defined in [RFC7277].

The following is an example of envisioned usage. The interfaces container includes a number of commonly used components as examples:

```

module: ietf-interfaces
  +--rw interfaces
  |   +--rw interface* [name]
  |   |   +--rw name                                string
  |   |   +--rw ip:ipv4!
  |   |   |   +--rw ip:enabled?                      boolean
  |   |   |   +--rw ip:forwarding?                  boolean
  |   |   |   +--rw ip:mtu?                          uint16
  |   |   |   +--rw ip:address* [ip]
  |   |   |   |   +--rw ip:ip                        inet:ipv4-address-no-zone
  |   |   |   |   +--rw (ip:subnet)
  |   |   |   |   |   +--:(ip:prefix-length)
  |   |   |   |   |   |   +--rw ip:prefix-length?   uint8
  |   |   |   |   |   +--:(ip:netmask)
  |   |   |   |   |   |   +--rw ip:netmask?         yang:dotted-quad
  |   |   |   +--rw ip:neighbor* [ip]
  |   |   |   |   +--rw ip:ip                        inet:ipv4-address-no-zone
  |   |   |   |   +--rw ip:link-layer-address       yang:phys-address
  |   |   |   +--rw ni:bind-network-instance-name?  string
  |   +--rw ni:bind-network-instance-name?         string

```

The [RFC7223] defined interface model is structured to include all interfaces in a flat list, without regard to virtual instances (e.g., VRFs) supported on the device. The bind-network-instance-name leaf provides the association between an interface and its associated NI (e.g., VRF or VSI). Note that as currently defined, to assign an interface to both an LNE and NI, the interface would first be assigned to the LNE using the mechanisms defined in [I-D.ietf-rtgwg-lne-model] and then within that LNE's interface module, the LNE's representation of that interface would be assigned to an NI.

3.3. Network Instance Management

Modules that may be used to represent network instance information will be available under the ni-type specific 'root' mount point. The use-schema mechanism defined as part of the Schema Mount module [I-D.ietf-netmod-schema-mount] MUST be used with the module defined in this document to identify accessible modules. A future version of this document could relax this requirement. Mounted modules in the non-inline case SHOULD be defined with access, via the appropriate schema mount parent-references [I-D.ietf-netmod-schema-mount], to device resources such as interfaces.

All modules that represent control-plane and data-plane information may be present at the 'root' mount point, and be accessible via paths modified per [I-D.ietf-netmod-schema-mount]. The list of available modules is expected to be implementation dependent, as is the method used by an implementation to support NIs.

For example, the following could be used to define the data organization of the example NI shown in Section 3.1.2:

```

"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "name": "vrf-root",
      "use-schema": [
        {
          "name": "ni-schema",
          "parent-reference": [
            "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"
          ]
        }
      ]
    }
  ],
  "schema": [
    {
      "name": "ni-schema",
      "module": [
        {
          "name": "ietf-routing",
          "revision": "2016-11-04",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-routing",
          "conformance-type": "implement"
        },
        {
          "name": "ietf-ospf",
          "revision": "2017-03-12",
          "namespace":
            "urn:ietf:params:xml:ns:yang:ietf-ospf",
          "conformance-type": "implement"
        }
      ]
    }
  ]
}

```

Module data identified under "schema" will be instantiated under the mount point identified under "mount-point". These modules will be able to reference information for nodes belonging to top-level modules that are identified under "parent-reference". Parent referenced information is available to clients via their top level paths only, and not under the associated mount point.

3.4. Network Instance Instantiation

Network instances may be controlled by clients using existing list operations. When a list entry is created, a new instance is instantiated. The models mounted under an NI root are expected to be dependent on the server implementation. When a list entry is deleted, an existing network instance is destroyed. For more information, see [RFC7950] Section 7.8.6.

Once instantiated, host network device resources can be associated with the new NI. As previously mentioned, this document augments ietf-interfaces with the bind-ni-name leaf to support such associations for interfaces. When a bind-ni-name is set to a valid NI name, an implementation MUST take whatever steps are internally necessary to assign the interface to the NI or provide an error message (defined below) with an indication of why the assignment failed. It is possible for the assignment to fail while processing the set operation, or after asynchronous processing. Error notification in the latter case is supported via a notification.

4. Security Considerations

There are two different sets of security considerations to consider in the context of this document. One set is security related to information contained within mounted modules. The security considerations for mounted modules are not substantively changed based on the information being accessible within the context of an NI. For example, when considering the modules defined in [RFC8022], the security considerations identified in that document are equally applicable, whether those modules are accessed at a server's root or under an NI instance's root node.

The second area for consideration is information contained in the NI module itself. NI information represents network configuration and route distribution policy information. As such, the security of this information is important, but it is fundamentally no different than any other interface or routing configuration information that has already been covered in [RFC7223] and [RFC8022].

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

/network-instances/network-instance: This list specifies the network instances and the related control plane protocols configured on a device.

/if:interfaces/if:interface/*/bind-network-instance-name: This leaf indicates the NI instance to which an interface is assigned.

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

5. 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-network-instance

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-network-instance
namespace: urn:ietf:params:xml:ns:yang:ietf-network-instance
prefix:    ni
reference:  RFC XXXX
```

6. Network Instance Model

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

```
<CODE BEGINS> file "ietf-network-instance@2017-07-03.yang"
module ietf-network-instance {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-instance";
  prefix ni;

  // import some basic types

  import ietf-interfaces {
    prefix if;
    reference "RFC 7223: A YANG Data Model for Interface
              Management";
  }
  import ietf-ip {
    prefix ip;
    reference "RFC 7277: A YANG Data Model for IP Management";
  }
  import ietf-yang-schema-mount {
```

```
    prefix yangmnt;
    reference "draft-ietf-netmod-schema-mount: YANG Schema Mount";
    // RFC Ed.: Please replace this draft name with the
    // corresponding RFC number
}

organization
  "IETF Routing Area (rtgwg) Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
  WG List:    <mailto:rtgwg@ietf.org>

  Author:     Lou Berger
               <mailto:lberger@labn.net>
  Author:     Christan Hopps
               <mailto:chopps@chopps.org>
  Author:     Acee Lindem
               <mailto:acee@cisco.com>
  Author:     Dean Bogdanovic
               <mailto:ivandean@gmail.com>";
description
  "This module is used to support multiple network instances
  within a single physical or virtual device. Network
  instances are commonly known as VRFs (virtual routing
  and forwarding) and VSIs (virtual switching instances).

  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.";

// RFC Ed.: replace XXXX with actual RFC number and remove
// this note
// RFC Ed.: please update TBD

revision 2017-07-02 {
  description
    "Initial revision.";
  reference "RFC TBD";
}
```

```
// top level device definition statements

container network-instances {
  description
    "Network instances each of which consists of a
    VRFs (virtual routing and forwarding) and/or
    VSIs (virtual switching instances).";
  reference "RFC 8022 - A YANG Data Model for Routing
    Management";
  list network-instance {
    key "name";
    description
      "List of network-instances.";
    leaf name {
      type string;
      description
        "device scoped identifier for the network
        instance.";
    }
    leaf enabled {
      type boolean;
      default "true";
      description
        "Flag indicating whether or not the network
        instance is enabled.";
    }
    leaf description {
      type string;
      description
        "Description of the network instance
        and its intended purpose.";
    }
  }
  choice ni-type {
    description
      "This node serves as an anchor point for different types
      of network instances. Each 'case' is expected to
      differ in terms of the information needed in the
      parent/core to support the NI, and may differ in their
      mounted schema definition. When the mounted schema is
      not expected to be the same for a specific type of NI
      a mount point should be defined.";
  }
  choice root-type {
    description
      "Well known mount points.";
    yangmnt:mount-point "vrf-root" {
      description
        "Root for L3VPN type models. This will typically
```

```
        not be an inline type mount point.";
    }
    yangmnt:mount-point "vsi-root" {
        description
            "Root for L2VPN type models. This will typically
             not be an inline type mount point.";
    }
    yangmnt:mount-point "vv-root" {
        description
            "Root models that support both L2VPN type bridging
             and L3VPN type routing. This will typically
             not be an inline type mount point.";
    }
}
}
}

// augment statements

augment "/if:interfaces/if:interface" {
    description
        "Add a node for the identification of the network
         instance associated with the information configured
         on a interface.

        Note that a standard error will be returned if the
        identified leafref isn't present. If an interfaces cannot
        be assigned for any other reason, the operation SHALL fail
        with an error-tag of 'operation-failed' and an
        error-app-tag of 'ni-assignment-failed'. A meaningful
        error-info that indicates the source of the assignment
        failure SHOULD also be provided.";
    leaf bind-ni-name {
        type leafref {
            path "/network-instances/network-instance/name";
        }
        description
            "Network Instance to which an interface is bound.";
    }
}
}
augment "/if:interfaces/if:interface/ip:ipv4" {
    description
        "Add a node for the identification of the network
         instance associated with the information configured
         on an IPv4 interface.

        Note that a standard error will be returned if the
        identified leafref isn't present. If an interfaces cannot
```

```
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
    }
    description
      "Network Instance to which IPv4 interface is bound.";
  }
}
augment "/if:interfaces/if:interface/ip:ipv6" {
  description
    "Add a node for the identification of the network
    instance associated with the information configured
    on an IPv6 interface.

    Note that a standard error will be returned if the
    identified leafref isn't present. If an interfaces cannot
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
    }
    description
      "Network Instance to which IPv6 interface is bound.";
  }
}

// notification statements

notification bind-ni-name-failed {
  description
    "Indicates an error in the association of an interface to an
    NI. Only generated after success is initially returned when
    bind-ni-name is set.

    Note: some errors may need to be reported for multiple
    associations, e.g., a single error may need to be reported
    for an IPv4 and an IPv6 bind-ni-name.

    At least one container with a bind-ni-name leaf MUST be
```

```
        included in this notification.";
    leaf name {
        type leafref {
            path "/if:interfaces/if:interface/if:name";
        }
        mandatory true;
        description
            "Contains the interface name associated with the
            failure.";
    }
    container interface {
        description
            "Generic interface type.";
        leaf bind-ni-name {
            type leafref {
                path "/if:interfaces/if:interface/ni:bind-ni-name";
            }
            description
                "Contains the bind-ni-name associated with the
                failure.";
        }
    }
    container ipv4 {
        description
            "IPv4 interface type.";
        leaf bind-ni-name {
            type leafref {
                path "/if:interfaces/if:interface"
                    + "/ip:ipv4/ni:bind-ni-name";
            }
            description
                "Contains the bind-ni-name associated with the
                failure.";
        }
    }
    container ipv6 {
        description
            "IPv6 interface type.";
        leaf bind-ni-name {
            type leafref {
                path "/if:interfaces/if:interface"
                    + "/ip:ipv6/ni:bind-ni-name";
            }
            description
                "Contains the bind-ni-name associated with the
                failure.";
        }
    }
}
```

```
    leaf error-info {  
        type string;  
        description  
            "Optionally, indicates the source of the assignment  
            failure.";  
    }  
}  
}  
<CODE ENDS>
```

7. References

7.1. Normative References

- [I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-05 (work in progress), May 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.
- [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>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<http://www.rfc-editor.org/info/rfc7277>>.

7.2. Informative References

- [I-D.ietf-bess-l2vpn-yang]
Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B.,
and K. Tiruveedhula, "YANG Data Model for MPLS-based
L2VPN", draft-ietf-bess-l2vpn-yang-05 (work in progress),
March 2017.
- [I-D.ietf-bess-l3vpn-yang]
Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S.,
Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model
for BGP/MPLS L3 VPNs", draft-ietf-bess-l3vpn-yang-01 (work
in progress), April 2017.
- [I-D.ietf-ospf-yang]
Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem,
"Yang Data Model for OSPF Protocol", draft-ietf-ospf-
yang-08 (work in progress), July 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., and D. Bogdanovic,
"YANG Logical Network Elements", draft-ietf-rtgwg-lne-
model-02 (work in progress), March 2017.
- [RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual
Private Network (VPN) Terminology", RFC 4026,
DOI 10.17487/RFC4026, March 2005,
<<http://www.rfc-editor.org/info/rfc4026>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private
Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February
2006, <<http://www.rfc-editor.org/info/rfc4364>>.
- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer
2 Virtual Private Networks (L2VPNs)", RFC 4664,
DOI 10.17487/RFC4664, September 2006,
<<http://www.rfc-editor.org/info/rfc4664>>.
- [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>>.

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <<http://www.rfc-editor.org/info/rfc8022>>.

Appendix A. Acknowledgments

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Yan Gang. Useful review comments were also received by Martin Bjorklund and John Scudder.

This document was motivated by, and derived from, [I-D.ietf-rtgwg-device-model].

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

Appendix B. Example NI usage

The following subsections provide example uses of NIs.

B.1. Configuration Data

The following shows an example where two customer specific network instances are configured:

```
{
  "ietf-network-instance:network-instances": {
    "network-instance": [
      {
        "name": "vrf-red",
        "vrf-root": {
          "ietf-routing:routing": {
            "router-id": "192.0.2.1",
            "control-plane-protocols": {
              "control-plane-protocol": [
                {
                  "type": "ietf-routing:ospf",
                  "name": "1",
                  "ietf-ospf:ospf": {
                    "instance": [
                      {
                        "af": "ipv4",
                        "areas": {
                          "area": [
                            {
                              "area-id": "203.0.113.1",
                              "interfaces": {
                                "interface": [
```

```

        {
            "name": "eth1",
            "cost": 10
        }
    ]
}
]
}
]
}
]
}
]
}
]
}
]
}
]
}
],
{
    "name": "vrf-blue",
    "vrf-root": {
        "ietf-routing:routing": {
            "router-id": "192.0.2.2",
            "control-plane-protocols": {
                "control-plane-protocol": [
                    {
                        "type": "ietf-routing:ospf",
                        "name": "1",
                        "ietf-ospf:ospf": {
                            "instance": [
                                {
                                    "af": "ipv4",
                                    "areas": {
                                        "area": [
                                            {
                                                "area-id": "203.0.113.1",
                                                "interfaces": {
                                                    "interface": [
                                                        {
                                                            "name": "eth2",
                                                            "cost": 10
                                                        }
                                                    ]
                                                }
                                            }
                                        ]
                                    }
                                }
                            ]
                        }
                    }
                ]
            }
        }
    }
}

```

```

    ]
  }
}
]
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        }
      },
      {
        "name": "eth1",
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      },
      {
        "ni:bind-network-instance-name": "vrf-red"
      },
      {
        "name": "eth2",
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      },
      {
        "ni:bind-network-instance-name": "vrf-blue"
      }
    ]
  }
}

```

```

    }
  ]
}
},
"ietf-system:system": {
  "authentication": {
    "user": [
      {
        "name": "john",
        "password": "$0$password"
      }
    ]
  }
}
}

```

B.2. State Data

The following shows state data for the example above.

```

{
  "ietf-network-instance:network-instances": {
    "network-instance": [
      {
        "name": "vrf-red",
        "vrf-root": {
          "ietf-routing:routing-state": {
            "router-id": "192.0.2.1",
            "control-plane-protocols": {
              "control-plane-protocol": [
                {
                  "type": "ietf-routing:ospf",
                  "name": "1",
                  "ietf-ospf:ospf": {
                    "instance": [
                      {
                        "af": "ipv4",
                        "areas": {
                          "area": [
                            {
                              "area-id": "203.0.113.1",
                              "interfaces": {
                                "interface": [
                                  {
                                    "name": "eth1",
                                    "cost": 10
                                  }
                                ]
                              }
                            }
                          ]
                        }
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
      }
    ]
  }
}

```

```

    },
    {
      "name": "vrf-blue",
      "vrf-root": {
        "ietf-routing:routing-state": {
          "router-id": "192.0.2.2",
          "control-plane-protocols": {
            "control-plane-protocol": [
              {
                "type": "ietf-routing:ospf",
                "name": "1",
                "ietf-ospf:ospf": {
                  "instance": [
                    {
                      "af": "ipv4",
                      "areas": {
                        "area": [
                          {
                            "area-id": "203.0.113.1",
                            "interfaces": {
                              "interface": [
                                {
                                  "name": "eth2",
                                  "cost": 10
                                }
                              ]
                            }
                          }
                        ]
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
      }
    }
  ],
  ]
}

```

```

    }
  }
}
],
},
"ietf-interfaces:interfaces-state": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C0",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        }
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",
              "prefix-length": 24,
            }
          ]
        }
      },
      {
        "name": "eth2",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",

```

```
    "phys-address": "00:01:02:A1:B1:C2",
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    },
    "ip:ipv4": {
      "address": [
        {
          "ip": "192.0.2.11",
          "prefix-length": 24,
        }
      ]
    }
  }
}
},
"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "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-network-instance",  
    "revision": "2017-03-13",  
    "feature": [  
      "bind-network-instance-name"  
    ],  
    "namespace":  
    "urn:ietf:params:xml:ns:yang:ietf-network-instance",  
    "conformance-type": "implement"  
  },  
  {  
    "name": "ietf-ospf",  
    "revision": "2017-03-12",  
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-ospf",  
    "conformance-type": "implement"  
  },  
  {  
    "name": "ietf-routing",  
    "revision": "2016-11-04",  
    "namespace":  
    "urn:ietf:params:xml:ns:yang:ietf-routing",  
    "conformance-type": "implement"  
  },  
  {  
    "name": "ietf-system",  
    "revision": "2014-08-06",  
    "namespace":  
    "urn:ietf:params:xml:ns:yang:ietf-system",  
    "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"
}
],
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
}
}
```

Authors' Addresses

Lou Berger
LabN Consulting, L.L.C.
Email: lberger@labn.net

Christan Hopps
Deutsche Telekom
Email: chopps@chopps.org

Acee Lindem
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA
Email: acee@cisco.com

Dean Bogdanovic
Email: ivandean@gmail.com

Xufeng Liu
Jabil

Email: Xufeng_Liu@jabil.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2018

L. Berger
LabN Consulting, L.L.C.
C. Hopps
Deutsche Telekom
A. Lindem
Cisco Systems
D. Bogdanovic

X. Liu
Jabil
March 19, 2018

YANG Model for Network Instances
draft-ietf-rtgwg-ni-model-12

Abstract

This document defines a network instance module. This module can be used to manage the virtual resource partitioning that may be present on a network device. Examples of common industry terms for virtual resource partitioning are Virtual Routing and Forwarding (VRF) instances and Virtual Switch Instances (VSIs).

The YANG model in this document conforms to the Network Management Datastore Architecture defined in I-D.ietf-netmod-revised-datastores.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 20, 2018.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Overview	4
3. Network Instances	5
3.1. NI Types and Mount Points	6
3.1.1. Well Known Mount Points	7
3.1.2. NI Type Example	8
3.2. NIs and Interfaces	9
3.3. Network Instance Management	10
3.4. Network Instance Instantiation	12
4. Security Considerations	13
5. IANA Considerations	14
6. Network Instance Model	14
7. References	20
7.1. Normative References	20
7.2. Informative References	22
Appendix A. Acknowledgments	23
Appendix B. Example NI usage	23
B.1. Configuration Data	23
B.2. State Data - Non-NMDA Version	27
B.3. State Data - NMDA Version	33
Authors' Addresses	42

1. Introduction

This document defines the second of two new modules that are defined to support the configuration and operation of network-devices that allow for the partitioning of resources from both, or either, management and networking perspectives. Both leverage the YANG functionality enabled by YANG Schema Mount [I-D.ietf-netmod-schema-mount].

The YANG model in this document conforms to the Network Management Datastore Architecture defined in the [I-D.ietf-netmod-revised-datastores].

The first form of resource partitioning provides a logical partitioning of a network device where each partition is separately managed as essentially an independent network element which is 'hosted' by the base network device. These hosted network elements are referred to as logical network elements, or LNEs, and are supported by the logical-network-element module defined in [I-D.ietf-rtgwg-lne-model]. That module is used to identify LNEs and associate resources from the network-device with each LNE. LNEs themselves are represented in YANG as independent network devices; each accessed independently. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric.

The second form, which is defined in this document, provides support for what is commonly referred to as Virtual Routing and Forwarding (VRF) instances as well as Virtual Switch Instances (VSI), see [RFC4026] and [RFC4664]. In this form of resource partitioning, multiple control plane and forwarding/bridging instances are provided by and managed via a single (physical or logical) network device. This form of resource partitioning is referred to as a Network Instance and is supported by the network-instance module defined below. Configuration and operation of each network-instance is always via the network device and the network-instance module.

One notable difference between the LNE model and the NI model is that the NI model provides a framework for VRF and VSI management. This document envisions the separate definition of VRF and VSI, i.e., L3 and L2 VPN, technology specific models. An example of such can be found in the emerging L3VPN model defined in [I-D.ietf-bess-l3vpn-yang] and the examples discussed below.

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] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with terms and concepts of YANG [RFC7950] and YANG Schema Mount [I-D.ietf-netmod-schema-mount].

This document uses the graphical representation of data models defined in [I-D.ietf-netmod-yang-tree-diagrams].

2. Overview

In this document, we consider network devices that support protocols and functions defined within the IETF, e.g, routers, firewalls, and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs) each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or logical router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a network instances (NIs). This breakdown is represented in Figure 1.

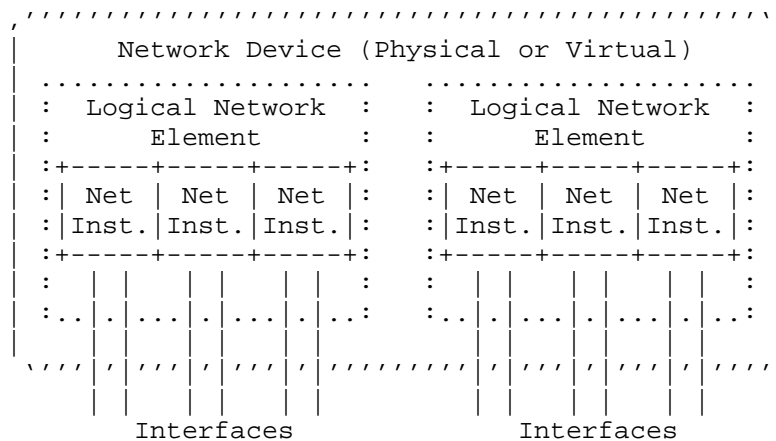


Figure 1: Module Element Relationships

A model for LNEs is described in [I-D.ietf-rtgwg-lne-model] and the model for NIs is covered in this document in Section 3.

The current interface management model [I-D.ietf-netmod-rfc7223bis] is impacted by the definition of LNEs and NIs. This document and [I-D.ietf-rtgwg-lne-model] define augmentations to the interface module to support LNEs and NIs.

The network instance model supports the configuration of VRFs and VSIs. Each instance is supported by information that relates to the device, for example the route target used when advertising VRF routes via the mechanisms defined in [RFC4364], and information that relates to the internal operation of the NI, for example for routing protocols [I-D.ietf-netmod-rfc8022bis] and OSPF [I-D.ietf-ospf-yang].

This document defines the network-instance module that provides a basis for the management of both types of information.

NI information that relates to the device, including the assignment of interfaces to NIs, is defined as part of this document. The defined module also provides a placeholder for the definition of NI-technology specific information both at the device level and for NI internal operation. Information related to NI internal operation is supported via schema mount [I-D.ietf-netmod-schema-mount] and mounting appropriate modules under the mount point. Well known mount points are defined for L3VPN, L2VPN, and L2+L3VPN NI types.

3. Network Instances

The network instance container is used to represent virtual routing and forwarding instances (VRFs) and virtual switching instances (VSIs). VRFs and VSIs are commonly used to isolate routing and switching domains, for example to create virtual private networks, each with their own active protocols and routing/switching policies. The model supports both core/provider and virtual instances. Core/provider instance information is accessible at the top level of the server, while virtual instance information is accessible under the root schema mount points.


```

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
        +--:(vsi-root)
          | +--mp vsi-root
        +--:(vv-root)
          +--mp vv-root
  augment /if:interfaces/if:interface:
    +--rw bind-ni-name? -> /network-instances/network-instance/name
  augment /if:interfaces/if:interface/ip:ipv4:
    +--rw bind-ni-name? -> /network-instances/network-instance/name
  augment /if:interfaces/if:interface/ip:ipv6:
    +--rw bind-ni-name? -> /network-instances/network-instance/name

notifications:
  +---n bind-ni-name-failed
    +--ro name                -> /if:interfaces/interface/name
    +--ro interface
      | +--ro bind-ni-name?
      |                               -> /if:interfaces/interface/ni:bind-ni-name
    +--ro ipv4
      | +--ro bind-ni-name?
      |                               -> /if:interfaces/interface/ip:ipv4/ni:bind-ni-name
    +--ro ipv6
      | +--ro bind-ni-name?
      |                               -> /if:interfaces/interface/ip:ipv6/ni:bind-ni-name
    +--ro error-info?        string

```

A network instance is identified by a 'name' string. This string is used both as an index within the network-instance module and to associate resources with a network instance as shown above in the interface augmentation. The ni-type and root-type choice statements are used to support different types of L2 and L3 VPN technologies. The bind-ni-name-failed notification is used in certain failure cases.

3.1. NI Types and Mount Points

The network-instance module is structured to facilitate the definition of information models for specific types of VRFs and VSIs using augmentations. For example, the information needed to support

VPLS, VxLAN and EVPN based L2VPNs are likely to be quite different. Example models under development that could be restructured to take advantage on NIs include, for L3VPNs [I-D.ietf-bess-l3vpn-yang] and for L2VPNs [I-D.ietf-bess-l2vpn-yang].

Documents defining new YANG models for the support of specific types of network instances should augment the network instance module. The basic structure that should be used for such augmentations include a case statement, with containers for configuration and state data and finally, when needed, a type specific mount point. Generally ni types, are expected to not need to define type specific mount points, but rather reuse one of the well known mount point, as defined in the next section. The following is an example type specific augmentation:

```
augment "/ni:network-instances/ni:network-instance/ni:ni-type" {
  case l3vpn {
    container l3vpn {
      ...
    }
    container l3vpn-state {
      ...
    }
  }
}
```

3.1.1. Well Known Mount Points

YANG Schema Mount, [I-D.ietf-netmod-schema-mount], identifies mount points by name within a module. This definition allows for the definition of mount points whose schema can be shared across ni-types. As discussed above, ni-types largely differ in the configuration information needed in the core/top level instance to support the NI, rather than in the information represented within an NI. This allows the use of shared mount points across certain NI types.

The expectation is that there are actually very few different schema that need to be defined to support NIs on an implementation. In particular, it is expected that the following three forms of NI schema are needed, and each can be defined with a well known mount point that can be reused by future modules defining ni-types.

The three well known mount points are:

vrf-root
vrf-root is intended for use with L3VPN type ni-types.

vsi-root

vsi-root is intended for use with L2VPN type ni-types.

vv-root

vv-root is intended for use with ni-types that simultaneously support L2VPN bridging and L3VPN routing capabilities.

Future model definitions should use the above mount points whenever possible. When a well known mount point isn't appropriate, a model may define a type specific mount point via augmentation.

3.1.2. NI Type Example

The following is an example of an L3VPN VRF using a hypothetical augmentation to the networking instance schema defined in [I-D.ietf-bess-l3vpn-yang]. More detailed examples can be found in Appendix B.

```

module: ietf-network-instance
  +--rw network-instances
    +--rw network-instance* [name]
      +--rw name string
      +--rw enabled? boolean
      +--rw description? string
      +--rw (ni-type)?
        | +--:(l3vpn)
        |   +--rw l3vpn:l3vpn
        |   |   ... // config data
        |   +--ro l3vpn:l3vpn-state
        |   |   ... // state data
        +--rw (root-type)
          +--:(vrf-root)
            +--mp vrf-root
              +--rw rt:routing/
                +--rw router-id? yang:dotted-quad
                +--rw control-plane-protocols
                  +--rw control-plane-protocol* [type name]
                  +--rw ospf:ospf/
                    +--rw area* [area-id]
                    +--rw interfaces
                      +--rw interface* [name]
                      +--rw name if:interface-ref
                      +--rw cost? uint16
                +--ro if:interfaces@
                | ...

```

This shows YANG Routing Management [I-D.ietf-netmod-rfc8022bis] and YANG OSPF [I-D.ietf-ospf-yang] as mounted modules. The mounted

modules can reference interface information via a parent-reference to the containers defined in [I-D.ietf-netmod-rfc7223bis].

3.2. NIs and Interfaces

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [I-D.ietf-netmod-rfc7223bis].

As shown below, the network-instance module augments the existing interface management model by adding a name which is used on interface or sub-interface types to identify an associated network instance. Similarly, this name is also added for IPv4 and IPv6 types, as defined in [I-D.ietf-netmod-rfc7277bis].

The following is an example of envisioned usage. The interfaces container includes a number of commonly used components as examples:

```
module: ietf-interfaces
  +--rw interfaces
  |   +--rw interface* [name]
  |   |   +--rw name                                string
  |   |   +--rw ip:ipv4!
  |   |   |   +--rw ip:enabled?                      boolean
  |   |   |   +--rw ip:forwarding?                   boolean
  |   |   |   +--rw ip:mtu?                          uint16
  |   |   |   +--rw ip:address* [ip]
  |   |   |   |   +--rw ip:ip                        inet:ipv4-address-no-zone
  |   |   |   |   +--rw (ip:subnet)
  |   |   |   |   |   +--:(ip:prefix-length)
  |   |   |   |   |   |   +--rw ip:prefix-length?    uint8
  |   |   |   |   |   +--:(ip:netmask)
  |   |   |   |   |   |   +--rw ip:netmask?          yang:dotted-quad
  |   |   |   +--rw ip:neighbor* [ip]
  |   |   |   |   +--rw ip:ip                        inet:ipv4-address-no-zone
  |   |   |   |   +--rw ip:link-layer-address        yang:phys-address
  |   |   |   +--rw ni:bind-network-instance-name?  string
  |   +--rw ni:bind-network-instance-name?          string
```

The [I-D.ietf-netmod-rfc7223bis] defined interface model is structured to include all interfaces in a flat list, without regard to virtual instances (e.g., VRFs) supported on the device. The bind-

network-instance-name leaf provides the association between an interface and its associated NI (e.g., VRF or VSI). Note that as currently defined, to assign an interface to both an LNE and NI, the interface would first be assigned to the LNE using the mechanisms defined in [I-D.ietf-rtgwg-lne-model] and then within that LNE's interface module, the LNE's representation of that interface would be assigned to an NI.

3.3. Network Instance Management

Modules that may be used to represent network instance information will be available under the ni-type specific 'root' mount point. The "shared-schema" method defined in the "ietf-yang-schema-mount" module [I-D.ietf-netmod-schema-mount] MUST be used to identify accessible modules. A future version of this document could relax this requirement. Mounted modules SHOULD be defined with access, via the appropriate schema mount parent-references [I-D.ietf-netmod-schema-mount], to device resources such as interfaces. An implementation MAY choose to restrict parent referenced information to information related to a specific instance, e.g., only allowing references to interfaces that have a "bind-network-instance-name" which is identical to the instance's "name".

All modules that represent control-plane and data-plane information may be present at the 'root' mount point, and be accessible via paths modified per [I-D.ietf-netmod-schema-mount]. The list of available modules is expected to be implementation dependent, as is the method used by an implementation to support NIs.

For example, the following could be used to define the data organization of the example NI shown in Section 3.1.2:

```
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "vrf-root",
      "shared-schema": {
        "parent-reference": [
          "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"
        ]
      }
    ]
  }
}
```

Module data identified according to the ietf-yang-schema-mount module will be instantiated under the mount point identified under "mount-

point". These modules will be able to reference information for nodes belonging to top-level modules that are identified under "parent-reference". Parent referenced information is available to clients via their top level paths only, and not under the associated mount point.

To allow a client to understand the previously mentioned instance restrictions on parent referenced information, an implementation MAY represent such restrictions in the "parent-reference" leaf-list. For example:

```
"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": "vrf-root",
    "shared-schema": {
      "parent-reference": [
        "/if:interfaces/if:interface
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces/if:interface/ip:ipv4
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces/if:interface/ip:ipv6
          [ni:bind-network-instance-name = current()/../ni:name]"
      ]
    }
  }
],
```

The same such "parent-reference" restrictions for non-NMDA implementations can be represented based on the [RFC7223] and [RFC7277] as:

```

"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": "vrf-root",
    "shared-schema": {
      "parent-reference": [
        "/if:interfaces/if:interface
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces-state/if:interface
          [if:name = /if:interfaces/if:interface
            [ni:bind-ni-name = current()/../ni:name]/if:name]",
        "/if:interfaces/if:interface/ip:ipv4
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces-state/if:interface/ip:ipv4
          [if:name = /if:interfaces/if:interface/ip:ipv4
            [ni:bind-ni-name = current()/../ni:name]/if:name]",
        "/if:interfaces/if:interface/ip:ipv6
          [ni:bind-network-instance-name = current()/../ni:name]",
        "/if:interfaces-state/if:interface/ip:ipv6
          [if:name = /if:interfaces/if:interface/ip:ipv4
            [ni:bind-ni-name = current()/../ni:name]/if:name]"
      ]
    }
  }
],

```

3.4. Network Instance Instantiation

Network instances may be controlled by clients using existing list operations. When a list entry is created, a new instance is instantiated. The models mounted under an NI root are expected to be dependent on the server implementation. When a list entry is deleted, an existing network instance is destroyed. For more information, see [RFC7950] Section 7.8.6.

Once instantiated, host network device resources can be associated with the new NI. As previously mentioned, this document augments ietf-interfaces with the bind-ni-name leaf to support such

associations for interfaces. When a bind-ni-name is set to a valid NI name, an implementation MUST take whatever steps are internally necessary to assign the interface to the NI or provide an error message (defined below) with an indication of why the assignment failed. It is possible for the assignment to fail while processing the set operation, or after asynchronous processing. Error notification in the latter case is supported via a notification.

4. 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 two different sets of security considerations to consider in the context of this document. One set is security related to information contained within mounted modules. The security considerations for mounted modules are not substantively changed based on the information being accessible within the context of an NI. For example, when considering the modules defined in [I-D.ietf-netmod-rfc8022bis], the security considerations identified in that document are equally applicable, whether those modules are accessed at a server's root or under an NI instance's root node.

The second area for consideration is information contained in the NI module itself. NI information represents network configuration and route distribution policy information. As such, the security of this information is important, but it is fundamentally no different than any other interface or routing configuration information that has already been covered in [I-D.ietf-netmod-rfc7223bis] and [I-D.ietf-netmod-rfc8022bis].

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

/network-instances/network-instance: This list specifies the network instances and the related control plane protocols configured on a device.

/if:interfaces/if:interface/*/bind-network-instance-name: This leaf indicates the NI instance to which an interface is assigned.

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

5. 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-network-instance

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-network-instance
namespace: urn:ietf:params:xml:ns:yang:ietf-network-instance
prefix: ni
reference: RFC XXXX

6. Network Instance Model

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

```
<CODE BEGINS> file "ietf-network-instance@2018-03-20.yang"
module ietf-network-instance {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-instance";
  prefix ni;

  // import some basic types

  import ietf-interfaces {
    prefix if;
    reference "draft-ietf-netmod-rfc7223bis: A YANG Data Model
              for Interface Management";
  }
  import ietf-ip {
    prefix ip;
```

```
reference "draft-ietf-netmod-rfc7277bis: A YANG Data Model
        for IP Management";
}
import ietf-yang-schema-mount {
  prefix yangmnt;
  reference "draft-ietf-netmod-schema-mount: YANG Schema Mount";
  // RFC Ed.: Please replace this draft name with the
  // corresponding RFC number
}

organization
  "IETF Routing Area (rtgwg) Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
   WG List:   <mailto:rtgwg@ietf.org>

   Author:    Lou Berger
               <mailto:lberger@labn.net>
   Author:    Christan Hopps
               <mailto:chopps@chopps.org>
   Author:    Acee Lindem
               <mailto:acee@cisco.com>
   Author:    Dean Bogdanovic
               <mailto:ivandean@gmail.com>";

description
  "This module is used to support multiple network instances
   within a single physical or virtual device.  Network
   instances are commonly known as VRFs (virtual routing
   and forwarding) and VSIs (virtual switching instances).

   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.";

// RFC Ed.: replace XXXX with actual RFC number and remove
// this note
// RFC Ed.: please update TBD

revision 2018-03-20 {
```

```
    description
      "Initial revision.";
    reference "RFC TBD";
  }

  // top level device definition statements

  container network-instances {
    description
      "Network instances each of which consists of a
       VRFs (virtual routing and forwarding) and/or
       VSIs (virtual switching instances).";
    reference "draft-ietf-rtgwg-rfc8022bis - A YANG Data Model
              for Routing Management";
    list network-instance {
      key "name";
      description
        "List of network-instances.";
      leaf name {
        type string;
        mandatory true;
        description
          "device scoped identifier for the network
           instance.";
      }
      leaf enabled {
        type boolean;
        default "true";
        description
          "Flag indicating whether or not the network
           instance is enabled.";
      }
      leaf description {
        type string;
        description
          "Description of the network instance
           and its intended purpose.";
      }
      choice ni-type {
        description
          "This node serves as an anchor point for different types
           of network instances. Each 'case' is expected to
           differ in terms of the information needed in the
           parent/core to support the NI, and may differ in their
           mounted schema definition. When the mounted schema is
           not expected to be the same for a specific type of NI
           a mount point should be defined.";
      }
    }
  }
```

```

choice root-type {
  mandatory true;
  description
    "Well known mount points.";
  container vrf-root {
    description
      "Container for mount point.";
    yangmnt:mount-point "vrf-root" {
      description
        "Root for L3VPN type models. This will typically
        not be an inline type mount point.";
    }
  }
  container vsi-root {
    description
      "Container for mount point.";
    yangmnt:mount-point "vsi-root" {
      description
        "Root for L2VPN type models. This will typically
        not be an inline type mount point.";
    }
  }
  container vv-root {
    description
      "Container for mount point.";
    yangmnt:mount-point "vv-root" {
      description
        "Root models that support both L2VPN type bridging
        and L3VPN type routing. This will typically
        not be an inline type mount point.";
    }
  }
}
}
}

// augment statements

augment "/if:interfaces/if:interface" {
  description
    "Add a node for the identification of the network
    instance associated with the information configured
    on a interface.

    Note that a standard error will be returned if the
    identified leafref isn't present. If an interfaces cannot
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an

```

```
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
    }
    description
      "Network Instance to which an interface is bound.";
  }
}
augment "/if:interfaces/if:interface/ip:ipv4" {
  description
    "Add a node for the identification of the network
    instance associated with the information configured
    on an IPv4 interface.

    Note that a standard error will be returned if the
    identified leafref isn't present. If an interfaces cannot
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
    }
    description
      "Network Instance to which IPv4 interface is bound.";
  }
}
augment "/if:interfaces/if:interface/ip:ipv6" {
  description
    "Add a node for the identification of the network
    instance associated with the information configured
    on an IPv6 interface.

    Note that a standard error will be returned if the
    identified leafref isn't present. If an interfaces cannot
    be assigned for any other reason, the operation SHALL fail
    with an error-tag of 'operation-failed' and an
    error-app-tag of 'ni-assignment-failed'. A meaningful
    error-info that indicates the source of the assignment
    failure SHOULD also be provided.";
  leaf bind-ni-name {
    type leafref {
      path "/network-instances/network-instance/name";
```

```
    }
    description
      "Network Instance to which IPv6 interface is bound.";
  }
}

// notification statements

notification bind-ni-name-failed {
  description
    "Indicates an error in the association of an interface to an
    NI. Only generated after success is initially returned when
    bind-ni-name is set.

    Note: some errors may need to be reported for multiple
    associations, e.g., a single error may need to be reported
    for an IPv4 and an IPv6 bind-ni-name.

    At least one container with a bind-ni-name leaf MUST be
    included in this notification.";
  leaf name {
    type leafref {
      path "/if:interfaces/if:interface/if:name";
    }
    mandatory true;
    description
      "Contains the interface name associated with the
      failure.";
  }
  container interface {
    description
      "Generic interface type.";
    leaf bind-ni-name {
      type leafref {
        path "/if:interfaces/if:interface/ni:bind-ni-name";
      }
      description
        "Contains the bind-ni-name associated with the
        failure.";
    }
  }
}
container ipv4 {
  description
    "IPv4 interface type.";
  leaf bind-ni-name {
    type leafref {
      path "/if:interfaces/if:interface"
        + "/ip:ipv4/ni:bind-ni-name";
    }
  }
}
```

```
    }
    description
      "Contains the bind-ni-name associated with the
       failure.";
  }
}
container ipv6 {
  description
    "IPv6 interface type.";
  leaf bind-ni-name {
    type leafref {
      path "/if:interfaces/if:interface"
        + "/ip:ipv6/ni:bind-ni-name";
    }
    description
      "Contains the bind-ni-name associated with the
       failure.";
  }
}
leaf error-info {
  type string;
  description
    "Optionally, indicates the source of the assignment
     failure.";
}
}
}
<CODE ENDS>
```

7. References

7.1. 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-10
(work in progress), January 2018.
- [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.

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

7.2. Informative References

- [I-D.ietf-bess-l2vpn-yang]
Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", draft-ietf-bess-l2vpn-yang-08 (work in progress), February 2018.
- [I-D.ietf-bess-l3vpn-yang]
Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S., Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model for BGP/MPLS L3 VPNs", draft-ietf-bess-l3vpn-yang-02 (work in progress), October 2017.
- [I-D.ietf-netmod-rfc8022bis]
Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", draft-ietf-netmod-rfc8022bis-11 (work in progress), January 2018.
- [I-D.ietf-ospf-yang]
Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "Yang Data Model for OSPF Protocol", draft-ietf-ospf-yang-10 (work in progress), March 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-09 (work in progress), March 2018.
- [RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, DOI 10.17487/RFC4026, March 2005, <<https://www.rfc-editor.org/info/rfc4026>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <<https://www.rfc-editor.org/info/rfc4364>>.

- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <<https://www.rfc-editor.org/info/rfc4664>>.
- [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>>.

Appendix A. Acknowledgments

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Yan Gang. Useful review comments were also received by Martin Bjorklund and John Scudder.

This document was motivated by, and derived from, [I-D.ietf-rtgwg-device-model].

Thanks for AD and IETF last call comments from Alia Atlas, Liang Xia, Benoit Claise, and Adam Roach.

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

Appendix B. Example NI usage

The following subsections provide example uses of NIs.

B.1. Configuration Data

The following shows an example where two customer specific network instances are configured:

```
{
  "ietf-network-instance:network-instances": {
    "network-instance": [
```

```

{
  "name": "vrf-red",
  "vrf-root": {
    "ietf-routing:routing": {
      "router-id": "192.0.2.1",
      "control-plane-protocols": {
        "control-plane-protocol": [
          {
            "type": "ietf-routing:ospf",
            "name": "1",
            "ietf-ospf:ospf": {
              "af": "ipv4",
              "areas": {
                "area": [
                  {
                    "area-id": "203.0.113.1",
                    "interfaces": {
                      "interface": [
                        {
                          "name": "eth1",
                          "cost": 10
                        }
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    }
  },
  {
    "name": "vrf-blue",
    "vrf-root": {
      "ietf-routing:routing": {
        "router-id": "192.0.2.2",
        "control-plane-protocols": {
          "control-plane-protocol": [
            {
              "type": "ietf-routing:ospf",
              "name": "1",
              "ietf-ospf:ospf": {
                "af": "ipv4",
                "areas": {
                  "area": [

```

```

    {
      "area-id": "203.0.113.1",
      "interfaces": {
        "interface": [
          {
            "name": "eth2",
            "cost": 10
          }
        ]
      }
    }
  ]
},
{
  "ietf-interfaces:interfaces": {
    "interfaces": {
      "interface": [
        {
          "name": "eth0",
          "ip:ipv4": {
            "address": [
              {
                "ip": "192.0.2.10",
                "prefix-length": 24,
              }
            ]
          },
          "ip:ipv6": {
            "address": [
              {
                "ip": "2001:db8:0:2::10",
                "prefix-length": 64,
              }
            ]
          }
        }
      ]
    },
    {
      "name": "eth1",
      "ip:ipv4": {

```

```

        "address": [
            {
                "ip": "192.0.2.11",
                "prefix-length": 24,
            }
        ],
    },
    "ip:ipv6": {
        "address": [
            {
                "ip": "2001:db8:0:2::11",
                "prefix-length": 64,
            }
        ]
    },
    "ni:bind-network-instance-name": "vrf-red"
},
{
    "name": "eth2",
    "ip:ipv4": {
        "address": [
            {
                "ip": "192.0.2.11",
                "prefix-length": 24,
            }
        ]
    },
    "ip:ipv6": {
        "address": [
            {
                "ip": "2001:db8:0:2::11",
                "prefix-length": 64,
            }
        ]
    },
    "ni:bind-network-instance-name": "vrf-blue"
}
]
}
},
"ietf-system:system": {
    "authentication": {
        "user": [
            {
                "name": "john",
                "password": "$0$password"
            }
        ]
    }
}

```

```

    }
  }
}

```

B.2. State Data - Non-NMDA Version

The following shows state data for the configuration example above based on [RFC7223], [RFC7277], and [RFC8022].

```

{
  "ietf-network-instance:network-instances": {
    "network-instance": [
      {
        "name": "vrf-red",
        "vrf-root": {
          "ietf-yang-library:modules-state": {
            "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
            "module": [
              {
                "name": "ietf-yang-library",
                "revision": "2016-06-21",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-library",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-ospf",
                "revision": "2018-03-03",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-ospf",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-routing",
                "revision": "2018-03-13",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-routing",
                "conformance-type": "implement"
              }
            ]
          },
          "ietf-routing:routing-state": {
            "router-id": "192.0.2.1",
            "control-plane-protocols": {
              "control-plane-protocol": [
                {
                  "type": "ietf-routing:ospf",

```

```

    "name": "1",
    "ietf-ospf:ospf": {
      "af": "ipv4",
      "areas": {
        "area": [
          {
            "area-id": "203.0.113.1",
            "interfaces": {
              "interface": [
                {
                  "name": "eth1",
                  "cost": 10
                }
              ]
            }
          }
        ]
      }
    }
  },
  {
    "name": "vrf-blue",
    "vrf-root": {
      "ietf-yang-library:modules-state": {
        "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
        "module": [
          {
            "name": "ietf-yang-library",
            "revision": "2016-06-21",
            "namespace":
              "urn:ietf:params:xml:ns:yang:ietf-yang-library",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-ospf",
            "revision": "2018-03-03",
            "namespace":
              "urn:ietf:params:xml:ns:yang:ietf-ospf",
            "conformance-type": "implement"
          },
          {
            "name": "ietf-routing",
            "revision": "2018-03-13",

```

```

        "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
        "conformance-type": "implement"
    }
]
},
"ietf-routing:routing-state": {
    "router-id": "192.0.2.2",
    "control-plane-protocols": {
        "control-plane-protocol": [
            {
                "type": "ietf-routing:ospf",
                "name": "1",
                "ietf-ospf:ospf": {
                    "af": "ipv4",
                    "areas": {
                        "area": [
                            {
                                "area-id": "203.0.113.1",
                                "interfaces": {
                                    "interface": [
                                        {
                                            "name": "eth2",
                                            "cost": 10
                                        }
                                    ]
                                }
                            }
                        ]
                    }
                }
            }
        ]
    }
}
},
"ietf-interfaces:interfaces-state": {
    "interfaces": {
        "interface": [
            {
                "name": "eth0",
                "type": "iana-if-type:ethernetCsmacd",
                "oper-status": "up",
                "phys-address": "00:01:02:A1:B1:C0",

```



```
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    },
    "ip:ipv4": {
      "address": [
        {
          "ip": "192.0.2.10",
          "prefix-length": 24,
        }
      ]
    }
    "ip:ipv6": {
      "address": [
        {
          "ip": "2001:db8:0:2::10",
          "prefix-length": 64,
        }
      ]
    }
  },
  {
    "name": "eth1",
    "type": "iana-if-type:ethernetCsmacd",
    "oper-status": "up",
    "phys-address": "00:01:02:A1:B1:C1",
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    },
    "ip:ipv4": {
      "address": [
        {
          "ip": "192.0.2.11",
          "prefix-length": 24,
        }
      ]
    }
    "ip:ipv6": {
      "address": [
        {
          "ip": "2001:db8:0:2::11",
          "prefix-length": 64,
        }
      ]
    }
  }
},
{
  "name": "eth2",
  "type": "iana-if-type:ethernetCsmacd",
```

```
    "oper-status": "up",
    "phys-address": "00:01:02:A1:B1:C2",
    "statistics": {
      "discontinuity-time": "2017-06-26T12:34:56-05:00"
    },
    "ip:ipv4": {
      "address": [
        {
          "ip": "192.0.2.11",
          "prefix-length": 24,
        }
      ]
    }
    "ip:ipv6": {
      "address": [
        {
          "ip": "2001:db8:0:2::11",
          "prefix-length": 64,
        }
      ]
    }
  }
}
},
"ietf-system:system-state": {
  "platform": {
    "os-name": "NetworkOS"
  }
}

"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "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-network-instance",
      "revision": "2018-02-03",
      "feature": [
        "bind-network-instance-name"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-network-instance",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ospf",
      "revision": "2018-03-03",
      "namespace": "urn:ietf:params:xml:ns:yang:ietf-ospf",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-routing",
      "revision": "2018-03-13",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-routing",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-system",
      "revision": "2014-08-06",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-system",
      "conformance-type": "implement"
    }
  ],
  "namespace":
    "urn:ietf:params:xml:ns:yang:ietf-system",
  "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"
    }
  ]
},
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "vrf-root",
      "shared-schema": {
        "parent-reference": [
          "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"
        ]
      }
    }
  ]
}
}

```

B.3. State Data - NMDA Version

The following shows state data for the configuration example above based on [I-D.ietf-netmod-rfc7223bis], [I-D.ietf-netmod-rfc7277bis], and [I-D.ietf-netmod-rfc8022bis].

```

{
  "ietf-network-instance:network-instances": {

```

```
"network-instance": [
  {
    "name": "vrf-red",
    "vrf-root": {
      "ietf-yang-library:yang-library": {
        "checksum": "41e2ab5dc325f6d86f743e8da3de323f1a61a801",
        "module-set": [
          {
            "name": "ni-modules",
            "module": [
              {
                "name": "ietf-yang-library",
                "revision": "2016-06-21",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-library",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-ospf",
                "revision": "2018-03-03",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-ospf",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-routing",
                "revision": "2018-03-13",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-routing",
                "conformance-type": "implement"
              }
            ]
          }
        ],
        "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"
          },
          {
            "name": "ietf-datastores",
            "revision": "2018-02-14",
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-datastores"
          }
        ]
      }
    }
  ]
}
```

```

    ]
  }
],
"schema": [
  {
    "name": "ni-schema",
    "module-set": [ "ni-modules" ]
  }
],
"datastore": [
  {
    "name": "ietf-datastores:running",
    "schema": "ni-schema"
  },
  {
    "name": "ietf-datastores:operational",
    "schema": "ni-schema"
  }
]
},
"ietf-routing:routing": {
  "router-id": "192.0.2.1",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
                "area-id": "203.0.113.1",
                "interfaces": {
                  "interface": [
                    {
                      "name": "eth1",
                      "cost": 10
                    }
                  ]
                }
              }
            ]
          }
        }
      }
    ]
  }
}

```

```

    }
  },
  {
    "name": "vrf-blue",
    "vrf-root": {
      "ietf-yang-library:yang-library": {
        "checksum": "41e2ab5dc325f6d86f743e8da3de323f1a61a801",
        "module-set": [
          {
            "name": "ni-modules",
            "module": [
              {
                "name": "ietf-yang-library",
                "revision": "2016-06-21",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-yang-library",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-ospf",
                "revision": "2018-03-03",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-ospf",
                "conformance-type": "implement"
              },
              {
                "name": "ietf-routing",
                "revision": "2018-03-13",
                "namespace":
                  "urn:ietf:params:xml:ns:yang:ietf-routing",
                "conformance-type": "implement"
              }
            ],
            "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"
              },
              {
                "name": "ietf-datastores",
                "revision": "2018-02-14",

```

```

        "namespace": "urn:ietf:params:xml:ns:yang:ietf-datastores"
      }
    ]
  },
  "schema": [
    {
      "name": "ni-schema",
      "module-set": [ "ni-modules" ]
    }
  ],
  "datastore": [
    {
      "name": "ietf-datastores:running",
      "schema": "ni-schema"
    },
    {
      "name": "ietf-datastores:operational",
      "schema": "ni-schema"
    }
  ]
},
"ietf-routing:routing": {
  "router-id": "192.0.2.2",
  "control-plane-protocols": {
    "control-plane-protocol": [
      {
        "type": "ietf-routing:ospf",
        "name": "1",
        "ietf-ospf:ospf": {
          "af": "ipv4",
          "areas": {
            "area": [
              {
                "area-id": "203.0.113.1",
                "interfaces": {
                  "interface": [
                    {
                      "name": "eth2",
                      "cost": 10
                    }
                  ]
                }
              }
            ]
          }
        }
      }
    ]
  }
}

```



```

    ]
  }
}
]
},
"ietf-interfaces:interfaces": {
  "interfaces": {
    "interface": [
      {
        "name": "eth0",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C0",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.10",
              "prefix-length": 24,
            }
          ]
        },
        "ip:ipv6": {
          "address": [
            {
              "ip": "2001:db8:0:2::10",
              "prefix-length": 64,
            }
          ]
        }
      },
      {
        "name": "eth1",
        "type": "iana-if-type:ethernetCsmacd",
        "oper-status": "up",
        "phys-address": "00:01:02:A1:B1:C1",
        "statistics": {
          "discontinuity-time": "2017-06-26T12:34:56-05:00"
        },
        "ip:ipv4": {
          "address": [
            {
              "ip": "192.0.2.11",

```

```

        "prefix-length": 24,
      }
    ]
  }
  "ip:ipv6": {
    "address": [
      {
        "ip": "2001:db8:0:2::11",
        "prefix-length": 64,
      }
    ]
  }
},
{
  "name": "eth2",
  "type": "iana-if-type:ethernetCsmacd",
  "oper-status": "up",
  "phys-address": "00:01:02:A1:B1:C2",
  "statistics": {
    "discontinuity-time": "2017-06-26T12:34:56-05:00"
  },
  "ip:ipv4": {
    "address": [
      {
        "ip": "192.0.2.11",
        "prefix-length": 24,
      }
    ]
  }
  "ip:ipv6": {
    "address": [
      {
        "ip": "2001:db8:0:2::11",
        "prefix-length": 64,
      }
    ]
  }
}
]
},
{
  "ietf-system:system-state": {
    "platform": {
      "os-name": "NetworkOS"
    }
  }
}

```

```
"ietf-yang-library:modules-state": {
  "module-set-id": "123e4567-e89b-12d3-a456-426655440000",
  "module": [
    {
      "name": "iana-if-type",
      "revision": "2014-05-08",
      "namespace":
        "urn:ietf:params:xml:ns:yang:iana-if-type",
      "conformance-type": "import"
    },
    {
      "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": "2018-01-09",
      "feature": [
        "arbitrary-names",
        "pre-provisioning"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-interfaces",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ip",
      "revision": "2018-01-09",
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-ip",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-network-instance",
      "revision": "2018-02-03",
      "feature": [
        "bind-network-instance-name"
      ],
      "namespace":
        "urn:ietf:params:xml:ns:yang:ietf-network-instance",
      "conformance-type": "implement"
    },
    {
      "name": "ietf-ospf",
      "revision": "2017-10-30",
```

```
    "namespace": "urn:ietf:params:xml:ns:yang:ietf-ospf",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-routing",
    "revision": "2018-01-25",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-routing",
    "conformance-type": "implement"
  },
  {
    "name": "ietf-system",
    "revision": "2014-08-06",
    "namespace":
      "urn:ietf:params:xml:ns:yang:ietf-system",
    "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"
  }
]
},
"ietf-yang-schema-mount:schema-mounts": {
  "mount-point": [
    {
      "module": "ietf-network-instance",
      "label": "vrf-root",
      "shared-schema": {
        "parent-reference": [
```

```
        "/*[namespace-uri() = 'urn:ietf:...:ietf-interfaces']"  
      ]  
    }  
  ]  
}
```

Authors' Addresses

Lou Berger
LabN Consulting, L.L.C.

Email: lberger@labn.net

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org

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

Email: acee@cisco.com

Dean Bogdanovic

Email: ivandean@gmail.com

Xufeng Liu
Jabil

Email: Xufeng_Liu@jabil.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2017

X. Liu
Jabil
Y. Qu
Futurewei Technologies, Inc.
A. Lindem
Cisco Systems
C. Hopps
Deutsche Telekom
L. Berger
LabN Consulting, L.L.C.
June 29, 2017

Routing Area Common YANG Data Types
draft-ietf-rtgwg-routing-types-08

Abstract

This document defines a collection of common data types using the YANG data modeling language. These derived common types are designed to be imported by other modules defined in the routing area.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 31, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	2
2. Overview	2
3. IETF Routing Types YANG Module	6
4. IANA Routing Types YANG Module	23
5. IANA Considerations	34
5.1. IANA-Maintained iana-routing-types Module	35
6. Security Considerations	36
7. Acknowledgements	36
8. References	36
8.1. Normative References	36
8.2. Informative References	37
Authors' Addresses	39

1. Introduction

The YANG [RFC6020] [RFC7950] is a data modeling language used to model configuration data, state data, Remote Procedure Calls, and notifications for network management protocols. The YANG language supports a small set of built-in data types and provides mechanisms to derive other types from the built-in types.

This document introduces a collection of common data types derived from the built-in YANG data types. The derived types are designed to be the common types applicable for modeling in the routing area.

1.1. Terminology

The terminology for describing YANG data models is found in [RFC7950].

2. Overview

This document defines the two models for common routing types, ietf-routing-types and iana-routing-types. The only module imports are from [RFC6991]. The ietf-routing-types model contains common routing types other than those corresponding directly to IANA mappings. These include:

router-id

Router Identifiers are commonly used to identify a nodes in routing and other control plane protocols. An example usage of router-id can be found in [I-D.ietf-ospf-yang].

route-target

Route Targets (RTs) are commonly used to control the distribution of virtual routing and forwarding (VRF) information, see [RFC4364], in support of virtual private networks (VPNs). An example usage can be found in [I-D.ietf-bess-l2vpn-yang].

ipv6-route-target

IPv6 Route Targets (RTs) are similar to standard Route Targets only they IPv6 Address Specific BGP Extended Communities as described in [RFC5701]. An IPv6 Route Target is 20 octets and includes an IPv6 address as the global administrator.

route-target-type

This type defines the import and export rules of Route Targets, as descibed in Section 4.3.1 of [RFC4364]. An example usage can be found in [I-D.ietf-idr-bgp-model].

route-distinguisher

Route Distinguishers (RDs) are commonly used to identify separate routes in support of virtual private networks (VPNs). For example, in [RFC4364], RDs are commonly used to identify independent VPNs and VRFs, and more generally, to identify multiple routes to the same prefix. An example usage can be found in [I-D.ietf-idr-bgp-model].

route-origin

Route Origin is commonly used to indicate the Site of Origin for Routng and forwarding (VRF) information, see [RFC4364], in support of virtual private networks (VPNs). An example usage can be found in [I-D.ietf-bess-l3vpn-yang].

ipv6-route-origin

An IPv6 Route Origin would also be used to indicate the Site of Origin for Routng and forwarding (VRF) information, see [RFC4364], in support of virtual private networks (VPNs). IPv6 Route Origins are IPv6 Address Specific BGP Extended Communities as described in [RFC5701]. An IPv6 Route Origin is 20 octets and includes an IPv6 address as the global administrator.

ipv4-multicast-group-address

This type defines the representation of an IPv4 multicast group address, which is in the range from 224.0.0.0 to 239.255.255.255. An example usage can be found in [I-D.ietf-pim-yang].

ipv6-multicast-group-address

This type defines the representation of an IPv6 multicast group address, which is in the range of FF00::/8. An example usage can be found in [I-D.ietf-pim-yang].

ip-multicast-group-address

This type represents an IP multicast group address and is IP version neutral. The format of the textual representation implies the IP version. An example usage can be found in [I-D.ietf-pim-yang].

ipv4-multicast-source-address

IPv4 source address type for use in multicast control protocols. This type also allows the indication of wildcard sources, i.e., "*". An example of where this type may/will be used is [I-D.ietf-pim-yang].

ipv6-multicast-source-address

IPv6 source address type for use in multicast control protocols. This type also allows the indication of wildcard sources, i.e., "*". An example of where this type may/will be used is [I-D.ietf-pim-yang].

bandwidth-ieee-float32

Bandwidth in IEEE 754 floating point 32-bit binary format [IEEE754]. Commonly used in Traffic Engineering control plane protocols. An example of where this type may/will be used is [I-D.ietf-ospf-yang].

link-access-type

This type identifies the IGP link type. An example of where this type may/will be used is [I-D.ietf-ospf-yang].

timer-multiplier

This type is used in conjunction with a timer-value type. It is generally used to indicate define the number of timer-value intervals that may expire before a specific event must occur. Examples of this include the arrival of any BFD packets, see [RFC5880] Section 6.8.4, or hello_interval in [RFC3209]. Example of where this type may/will be used is [I-D.ietf-idr-bgp-model] and [I-D.ietf-teas-yang-rsvp].

timer-value-seconds16

This type covers timers which can be set in seconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint16 (2 octets). An example of where this type may/will be used is [I-D.ietf-ospf-yang].

timer-value-seconds32

This type covers timers which can be set in seconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint32 (4 octets). An example of where this type may/will be used is [I-D.ietf-teas-yang-rsvp].

timer-value-milliseconds

This type covers timers which can be set in milliseconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint32 (4 octets). Examples of where this type may/will be used include [I-D.ietf-teas-yang-rsvp] and [I-D.ietf-bfd-yang].

percentage

This type defines a percentage with a range of 0-100%. An example usage can be found in [I-D.ietf-idr-bgp-model].

timeticks64

This type is based on the timeticks type defined in [RFC6991] but with 64-bit precision. It represents the time in hundredths of a second between two epochs. An example usage can be found in [I-D.ietf-idr-bgp-model].

uint24

This type defines a 24-bit unsigned integer. It is used by target="I-D.ietf-ospf-yang"/>.

generalized-label

This type represents a generalized label for Generalized Multi-Protocol Label Switching (GMPLS) [RFC3471]. The Generalized Label does not identify its type, which is known from the context. An example usage can be found in [I-D.ietf-teas-yang-te].

mpls-label-special-purpose

This type represents the special-purpose Multiprotocol Label Switching (MPLS) label values [RFC7274]. An example usage can be found in [I-D.ietf-mpls-base-yang].

mpls-label-general-use

The 20 bits label values in an MPLS label stack entry, specified in [RFC3032]. This label value does not include the encodings of Traffic Class and TTL (time to live). The label range specified by this type is for general use, with special-purpose MPLS label values excluded. An example usage can be found in [I-D.ietf-mpls-base-yang].

mpls-label

The 20 bits label values in an MPLS label stack entry, specified in [RFC3032]. This label value does not include the encodings of Traffic Class and TTL (time to live). The label range specified by this type covers the general use values and the special-purpose label values. An example usage can be found in [I-D.ietf-mpls-base-yang].

This document defines the following YANG groupings:

`mpls-label-stack`

This grouping defines a reusable collection of schema nodes representing an MPLS label stack [RFC3032]. An example usage can be found in [I-D.ietf-mpls-base-yang].

`vpn-route-targets`

This grouping defines a reusable collection of schema nodes representing Route Target import-export rules used in the BGP enabled Virtual Private Networks (VPNs). [RFC4364][RFC4664]. An example usage can be found in [I-D.ietf-bess-l2vpn-yang].

`geo-coordinates`

This grouping defines a reusable collection of schema nodes representing the Geo-coordinates in IETF models. The schema nodes specify the location of an object using the WGS-84 (World Geodetic System) reference coordinate system [WGS84]. This is expected to be used in augmentations to routing protocol models such as [I-D.ietf-ospf-yang].

The `iana-routing-types` model contains common routing types corresponding directly to IANA mappings. These include:

`address-family`

This type defines values for use in address family identifiers. The values are based on the IANA Address Family Numbers Registry [IANA-ADDRESS-FAMILY-REGISTRY]. An example usage can be found in [I-D.ietf-idr-bgp-model].

`subsequent-address-family`

This type defines values for use in subsequent address family (SAFI) identifiers. The values are based on the IANA Subsequent Address Family Identifiers (SAFI) Parameters Registry [IANA-SAFI-REGISTRY].

3. IETF Routing Types YANG Module

```
<CODE BEGINS> file "ietf-routing-types@2017-06-29.yang"
module iETF-routing-types {
  namespace "urn:ietf:params:xml:ns:yang:ietf-routing-types";
```

```
prefix rt-types;

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

organization
  "IETF RTGWG - Routing Area Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
   WG List:   <mailto:rtgwg@ietf.org>

   Editor:    Xufeng Lui
               <mailto:Xufeng_Lui@jabail.com>
               Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>
               Acee Lindem
               <mailto:acee@cisco.com>
               Christian Hopps
               <mailto:chopps@chopps.org>
               Lou Berger
               <mailto:lberger@labn.com>";
description
  "This module contains a collection of YANG data types
   considered generally useful for routing protocols.

   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 2017-06-29 {
  description
    "Initial revision.";
  reference "RFC TBD: Routing YANG Data Types";
}
```

```
/** Identities related to MPLS/GMPLS */

identity mpls-label-special-purpose-value {
  description
    "Base identity for deriving identities describing
    special-purpose Multiprotocol Label Switching (MPLS) label
    values.";
  reference
    "RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels.";
}

identity ipv4-explicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the IPv4 Explicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity router-alert-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Router Alert Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity ipv6-explicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the IPv6 Explicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity implicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Implicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity entropy-label-indicator {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Entropy Label Indicator.";
  reference
    "RFC6790: The Use of Entropy Labels in MPLS Forwarding.
    Sections 3 and 10.1.";
}
```

```
identity gal-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Generic Associated Channel
    Label (GAL).";
  reference
    "RFC5586: MPLS Generic Associated Channel.
    Sections 4 and 10.";
}

identity oam-alert-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the OAM Alert Label.";
  reference
    "RFC3429: Assignment of the 'OAM Alert Label' for
    Multiprotocol Label Switching Architecture (MPLS)
    Operation and Maintenance (OAM) Functions.
    Sections 3 and 6.";
}

identity extension-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Extension Label.";
  reference
    "RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels. Sections 3.1 and 5.";
}

/** Collection of types related to routing */

typedef router-id {
  type yang:dotted-quad;
  description
    "A 32-bit number in the dotted quad format assigned to each
    router. This number uniquely identifies the router within
    an Autonomous System.";
}

/** Collection of types related to VPN */

typedef route-target {
  type string {
    pattern
      '(0:(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3})|'
      + '[0-5]?\d{0,3}\d):(429496729[0-5]|42949672[0-8]\d|'
      + '4294967[01]\d{2}|429496[0-6]\d{3}|42949[0-5]\d{4})|'
```

```

+ '4294[0-8]\d{5}|429[0-3]\d{6}|42[0-8]\d{7}|4[01]\d{8}|'
+ '[0-3]?d{0,8}\d))|'
+ '(1:(((\d|[1-9]\d|1\d{2}|2[0-4]\d|25[0-5])\.)}{3}(\d|[1-9]\d|'
+ '1\d{2}|2[0-4]\d|25[0-5])):(6553[0-5]|655[0-2]\d|'
+ '65[0-4]\d{2}|6[0-4]\d{3}|[0-5]?d{0,3}\d))|'
+ '(2:(429496729[0-5]|42949672[0-8]\d|4294967[01]\d{2}|'
+ '429496[0-6]\d{3}|42949[0-5]\d{4}|4294[0-8]\d{5}|'
+ '429[0-3]\d{6}|42[0-8]\d{7}|4[01]\d{8}|[0-3]?d{0,8}\d):'
+ '(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
+ '[0-5]?d{0,3}\d))';
}
description
  "A route target is an 8-octet BGP extended community
  initially identifying a set of sites in a BGP
  VPN (RFC 4364). However, it has since taken on a more
  general role in BGP route filtering.
  A route target consists of three fields:
  a 2-octet type field, an administrator field,
  and an assigned number field.
  According to the data formats for type 0, 1, and 2 defined
  in RFC4360 and RFC5668, the encoding pattern is defined as:

  0:2-octet-asn:4-octet-number
  1:4-octet-ipv4addr:2-octet-number
  2:4-octet-asn:2-octet-number.

  Some valid examples are: 0:100:100, 1:1.1.1.1:100, and
  2:1234567890:203.";
reference
  "RFC4360: BGP Extended Communities Attribute.
  RFC5668: 4-Octet AS Specific BGP Extended Community.";
}

typedef ipv6-route-target {
  type string {
    pattern
      '(((|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}'
      + '((((|[0-9a-fA-F]{0,4}):)?(:[0-9a-fA-F]{0,4}))|'
      + '(((25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])\.){3}'
      + '(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])))'
      + ':'
      + '(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
      + '[0-5]?d{0,3}\d)';
    pattern '(((^[^:]+:){6}((^[^:]+:[^:]+)|(.*\..*)))|'
      + '(((^[^:]+:)*[^\:]+)??:((^[^:]+:)*[^\:]+)?))'
      + ':'
      + '(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
      + '[0-5]?d{0,3}\d)';
  }
}

```

```

    }
    description
      "An IPv6 route target is a 20-octet BGP IPv6 address
       specific extended community serving the same function
       as a standard 8-octet route target only allowing for
       an IPv6 address as the global administrator. The format
       is <ipv6-address:2-octet-number>.

       Some valid examples are: 2001:DB8::1:6544 and
       2001:DB8::5eb1:791:6b37:17958";
    reference
      "RFC5701: IPv6 Address Specific BGP Extended Community
       Attribute";
  }

  typedef route-target-type {
    type enumeration {
      enum "import" {
        value 0;
        description
          "The route target applies to route import.";
      }
      enum "export" {
        value 1;
        description
          "The route target applies to route export.";
      }
      enum "both" {
        value 2;
        description
          "The route target applies to both route import and
           route export.";
      }
    }
  }
  description
    "Indicates the role a route target takes
     in route filtering.";
  reference "RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs).";
}

typedef route-distinguisher {
  type string {
    pattern
      '(0:(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3})|'
      + '[0-5]? \d{0,3} \d):(429496729[0-5]|42949672[0-8]\d|'
      + '4294967[01]\d{2}|429496[0-6]\d{3}|42949[0-5]\d{4}|'
      + '4294[0-8]\d{5}|429[0-3]\d{6}|42[0-8]\d{7}|4[01]\d{8})|'
      + '[0-3]? \d{0,8} \d))|';
  }
}

```



```

+ '(1:(((\d|[1-9]\d|1\d{2}|2[0-4]\d|25[0-5])\.){3}(\d|[1-9]\d|
+ '1\d{2}|2[0-4]\d|25[0-5])):(6553[0-5]|655[0-2]\d|
+ '65[0-4]\d{2}|6[0-4]\d{3}|[0-5]?\d{0,3}\d))|'
+ '(2:(429496729[0-5]|42949672[0-8]\d|4294967[01]\d{2}|'
+ '429496[0-6]\d{3}|42949[0-5]\d{4}|4294[0-8]\d{5}|'
+ '429[0-3]\d{6}|42[0-8]\d{7}|4[01]\d{8}|[0-3]?\d{0,8}\d):'
+ '(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
+ '[0-5]?\d{0,3}\d))|'
+ '([3-9a-fA-F]|1-9a-fA-F)[\da-fA-F]{1,3}):'
+ '[\da-fA-F]{1,12})';
}
description
  "A route distinguisher is an 8-octet value used to distinguish
  routes from different BGP VPNs (RFC 4364). A route
  distinguisher consists of three fields: A 2-octet type field,
  an administrator field, and an assigned number field.
  According to the data formats for type 0, 1, and 2 defined in
  RFC4364, the encoding pattern is defined as:

  0:2-octet-asn:4-octet-number
  1:4-octet-ipv4addr:2-octet-number
  2:4-octet-asn:2-octet-number.
  2-octet-other-hex-number:6-octet-hex-number

  Some valid examples are: 0:100:100, 1:1.1.1.1:100, and
  2:1234567890:203.";
reference "RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs).";
}

typedef route-origin {
  type string {
    pattern
      '(0:(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
      + '[0-5]?\d{0,3}\d):(429496729[0-5]|42949672[0-8]\d|'
      + '4294967[01]\d{2}|429496[0-6]\d{3}|42949[0-5]\d{4}|'
      + '4294[0-8]\d{5}|429[0-3]\d{6}|42[0-8]\d{7}|4[01]\d{8}|'
      + '[0-3]?\d{0,8}\d))|'
      + '(1:(((\d|[1-9]\d|1\d{2}|2[0-4]\d|25[0-5])\.){3}(\d|[1-9]\d|'
      + '1\d{2}|2[0-4]\d|25[0-5])):(6553[0-5]|655[0-2]\d|'
      + '65[0-4]\d{2}|6[0-4]\d{3}|[0-5]?\d{0,3}\d))|'
      + '(2:(429496729[0-5]|42949672[0-8]\d|4294967[01]\d{2}|'
      + '429496[0-6]\d{3}|42949[0-5]\d{4}|4294[0-8]\d{5}|'
      + '429[0-3]\d{6}|42[0-8]\d{7}|4[01]\d{8}|[0-3]?\d{0,8}\d):'
      + '(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
      + '[0-5]?\d{0,3}\d))|'
      + '([3-9a-fA-F]|1-9a-fA-F)[\da-fA-F]{1,3}):'
      + '[\da-fA-F]{1,12})';
  }
}

```

```

description
  "A route origin is an 8-octet BGP extended community
  identifying the set of sites where the BGP route
  originated(RFC 4364). A route origin consists of three
  fields: A 2-octet type field, an administrator field,
  and an assigned number field. According to the data
  formats for type 0, 1, and 2 defined in RFC4364,
  the encoding pattern is defined as:

  0:2-octet-asn:4-octet-number
  1:4-octet-ipv4addr:2-octet-number
  2:4-octet-asn:2-octet-number.
  2-octet-other-hex-number:6-octet-hex-number

  Some valid examples are: 0:100:100, 1:1.1.1.1:100, and
  2:1234567890:203.";
reference
  "RFC4360: BGP Extended Communities Attribute.
  RFC4369: BGP/MPLS IP Virtual Private Networks (VPNs)
  RFC5668: 4-Octet AS Specific BGP Extended Community.";
}

typedef ipv6-route-origin {
  type string {
    pattern
      '((:[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}'
      + '(((([0-9a-fA-F]{0,4}):)?(:[0-9a-fA-F]{0,4}))|'
      + '(((25[0-5]|2[0-4][0-9]|01?[0-9]?[0-9])\.){3}'
      + '(25[0-5]|2[0-4][0-9]|01?[0-9]?[0-9])))'
      + ':'
      + '(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
      + '[0-5]?\d{0,3}\d)';
    pattern '(((([^:]+){6}(((^[^:]+:[^:]+)|(.*\..*)))|'
      + '(((([^:]+)*[^\:]+)?::((^[^:]+)*[^\:]+)?))'
      + ':'
      + '(6553[0-5]|655[0-2]\d|65[0-4]\d{2}|6[0-4]\d{3}|'
      + '[0-5]?\d{0,3}\d)';
  }
}
description
  "An IPv6 route origin is a 20-octet BGP IPv6 address
  specific extended community serving the same function
  as a standard 8-octet route only only allowing for
  an IPv6 address as the global administrator. The format
  is <ipv6-address:2-octet-number>.

  Some valid examples are: 2001:DB8::1:6544 and
  2001:DB8::5eb1:791:6b37:17958";
reference

```

```
    "RFC5701: IPv6 Address Specific BGP Extended Community
      Attribute";
  }

  /*** Collection of types common to multicast ***/

  typedef ipv4-multicast-group-address {
    type inet:ipv4-address {
      pattern '(2((2[4-9])|(3[0-9]))\.)\. *';
    }
    description
      "This type represents an IPv4 multicast group address,
       which is in the range from 224.0.0.0 to 239.255.255.255.";
    reference "RFC1112: Host Extensions for IP Multicasting.";
  }

  typedef ipv6-multicast-group-address {
    type inet:ipv6-address {
      pattern
        '([fF]{2}[0-9a-fA-F]{2}):\. *';
    }
    description
      "This type represents an IPv6 multicast group address,
       which is in the range of FF00::/8.";
    reference
      "RFC4291: IP Version 6 Addressing Architecture. Sec 2.7.
       RFC7346: IPv6 Multicast Address Scopes.";
  }

  typedef ip-multicast-group-address {
    type union {
      type ipv4-multicast-group-address;
      type ipv6-multicast-group-address;
    }
    description
      "This type represents a version-neutral IP multicast group
       address. The format of the textual representation implies
       the IP version.";
  }

  typedef ipv4-multicast-source-address {
    type union {
      type enumeration {
        enum "*" {
          description
            "Any source address.";
        }
      }
    }
  }
```

```

    type inet:ipv4-address;
  }
  description
    "Multicast source IPv4 address type.";
}

typedef ipv6-multicast-source-address {
  type union {
    type enumeration {
      enum "*" {
        description
          "Any source address.";
      }
    }
    type inet:ipv6-address;
  }
  description
    "Multicast source IPv6 address type.";
}

/** Collection of types common to protocols */

typedef bandwidth-ieee-float32 {
  type string {
    pattern
      '0[xX](0((\.\0)?)[pP](\+)?0?|(\.\0?))|'
      + '1(\.([\da-fA-F]{0,5}[02468aAcCeE]?))?[pP](\+)?(12[0-7]|'
      + '1[01]\d|0?\d?\d)?';
  }
  description
    "Bandwidth in IEEE 754 floating point 32-bit binary format:
    (-1)**(S) * 2**(Exponent-127) * (1 + Fraction),
    where Exponent uses 8 bits, and Fraction uses 23 bits.
    The units are octets per second.
    The encoding format is the external hexadecimal-significant
    character sequences specified in IEEE 754 and C99. The
    format is restricted to be normalized, non-negative, and
    non-fraction: 0x1.hhhhhhp{+}d or 0X1.HHHHHHP{+}D
    where 'h' and 'H' are hexadecimal digits, 'd' and 'D' are
    integers in the range of [0..127].
    When six hexadecimal digits are used for 'hhhhh' or 'HHHHH',
    the least significant digit must be an even number.
    'x' and 'X' indicate hexadecimal; 'p' and 'P' indicate power
    of two. Some examples are: 0x0p0, 0x1p10, and
    0x1.abcde2p+20";
  reference
    "IEEE Std 754-2008: IEEE Standard for Floating-Point
    Arithmetic.";
}

```

```
}

typedef link-access-type {
  type enumeration {
    enum "broadcast" {
      description
        "Specify broadcast multi-access network.";
    }
    enum "non-broadcast-multiaccess" {
      description
        "Specify Non-Broadcast Multi-Access (NBMA) network.";
    }
    enum "point-to-multipoint" {
      description
        "Specify point-to-multipoint network.";
    }
    enum "point-to-point" {
      description
        "Specify point-to-point network.";
    }
  }
  description
    "Link access type.";
}

typedef timer-multiplier {
  type uint8;
  description
    "The number of timer value intervals that should be
    interpreted as a failure.";
}

typedef timer-value-seconds16 {
  type union {
    type uint16 {
      range "1..65535";
    }
    type enumeration {
      enum "infinity" {
        description
          "The timer is set to infinity.";
      }
      enum "not-set" {
        description
          "The timer is not set.";
      }
    }
  }
}
```

```
    units "seconds";
    description
        "Timer value type, in seconds (16-bit range).";
}

typedef timer-value-seconds32 {
    type union {
        type uint32 {
            range "1..4294967295";
        }
        type enumeration {
            enum "infinity" {
                description
                    "The timer is set to infinity.";
            }
            enum "not-set" {
                description
                    "The timer is not set.";
            }
        }
    }
    units "seconds";
    description
        "Timer value type, in seconds (32-bit range).";
}

typedef timer-value-milliseconds {
    type union {
        type uint32 {
            range "1..4294967295";
        }
        type enumeration {
            enum "infinity" {
                description
                    "The timer is set to infinity.";
            }
            enum "not-set" {
                description
                    "The timer is not set.";
            }
        }
    }
    units "milliseconds";
    description
        "Timer value type, in milliseconds.";
}

typedef percentage {
```

```
    type uint8 {
      range "0..100";
    }
    description
      "Integer indicating a percentage value";
  }

  typedef timeticks64 {
    type uint64;
    description
      "This type is based on the timeticks type defined in
      RFC 6991, but with 64-bit width. It represents the time,
      modulo 2^64, in hundredths of a second between two epochs.";
    reference "RFC 6991 - Common YANG Data Types";
  }

  typedef uint24 {
    type uint32 {
      range "0 .. 16777215";
    }
    description
      "24-bit unsigned integer";
  }

  /*** Collection of types related to MPLS/GMPLS ***/

  typedef generalized-label {
    type binary;
    description
      "Generalized label. Nodes sending and receiving the
      Generalized Label are aware of the link-specific
      label context and type.";
    reference "RFC3471: Section 3.2";
  }

  typedef mpls-label-special-purpose {
    type identityref {
      base mpls-label-special-purpose-value;
    }
    description
      "This type represents the special-purpose Multiprotocol Label
      Switching (MPLS) label values.";
    reference
      "RFC3032: MPLS Label Stack Encoding.
      RFC7274: Allocating and Retiring Special-Purpose MPLS
      Labels.";
  }
```

```
typedef mpls-label-general-use {
  type uint32 {
    range "16..1048575";
  }
  description
    "The 20-bit label values in an MPLS label stack entry,
    specified in RFC3032. This label value does not include
    the encodings of Traffic Class and TTL (time to live).
    The label range specified by this type is for general use,
    with special-purpose MPLS label values excluded.";
  reference "RFC3032: MPLS Label Stack Encoding.";
}

typedef mpls-label {
  type union {
    type mpls-label-special-purpose;
    type mpls-label-general-use;
  }
  description
    "The 20-bit label values in an MPLS label stack entry,
    specified in RFC3032. This label value does not include
    the encodings of Traffic Class and TTL (time to live).";
  reference "RFC3032: MPLS Label Stack Encoding.";
}

/**** Groupings **/

grouping mpls-label-stack {
  description
    "A grouping that specifies an MPLS label stack.";
  container mpls-label-stack {
    description
      "Container for a list of MPLS label stack entries.";
    list entry {
      key "id";
      description
        "List of MPLS label stack entries.";
      leaf id {
        type uint8;
        description
          "Identifies the sequence of an MPLS label stack entries.
          An entry with smaller ID value is precedes an entry in
          the label stack with a smaller ID.";
      }
      leaf label {
        type rt-types:mpls-label;
        description
          "Label value.";
      }
    }
  }
}
```



```

    }
    leaf ttl {
        type uint8;
        description
            "Time to Live (TTL).";
        reference "RFC3032: MPLS Label Stack Encoding.";
    }
    leaf traffic-class {
        type uint8 {
            range "0..7";
        }
        description
            "Traffic Class (TC).";
        reference
            "RFC5462: Multiprotocol Label Switching (MPLS) Label
            Stack Entry: 'EXP' Field Renamed to 'Traffic Class'
            Field.";
    }
}
}
}

grouping vpn-route-targets {
    description
        "A grouping that specifies Route Target import-export rules
        used in the BGP enabled Virtual Private Networks (VPNs).";
    reference
        "RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs).
        RFC4664: Framework for Layer 2 Virtual Private Networks
        (L2VPNs)";
    list vpn-target {
        key "route-target";
        description
            "List of Route Targets.";
        leaf route-target {
            type rt-types:route-target;
            description
                "Route Target value";
        }
        leaf route-target-type {
            type rt-types:route-target-type;
            mandatory true;
            description
                "Import/export type of the Route Target.";
        }
    }
}
}

```

```
grouping geo-coordinates {
  description
    "Standard grouping for Geo Coordinates
    in routing information";
  container geo-coordinates {
    description
      "Container for Geo Coordinates";
    leaf flags {
      type bits {
        bit U {
          description
            "If the U-bit is set, it indicates that
            the location-uncertainty is specified, If the
            U-bit is clear, it indicates the
            location-uncertainty is unspecified.";
        }
        bit N {
          description
            "If the N-bit is set, it indicates the
            latitude is north relative to the Equator. If
            the N-bit is clear, it indicates the latitude
            is south of the Equator.";
        }
        bit E {
          description
            "If the E-bit is set, it indicates the
            longitude is east of the Prime Meridian. If
            the E-bit is clear, it indicates the longitude
            is west of the Prime Meridian.";
        }
        bit A {
          description
            "If the A-bit is set, it indicates the
            altitude is specified. If the A-bit is clear,
            it indicates the altitude is unspecified.";
        }
        bit M {
          description
            "If the M-bit is set, it indicates the
            altitude is specified in meters. If the M-bit
            is clear, it indicates the altitude is
            specified in centimeters.";
        }
        bit R {
          description
            "If the R-bit is set, it indicates the
            radius is specified and the encoding is for a
            circular area. If the R-bit is clear, it
```

```
        indicates the radius is unspecified and the
        encoding is for a single point.";
    }
    bit K {
        description
            "If the R-bit is set, it indicates the
            radius is specified in kilometers. If the
            R-bit is clear, it indicates the radius is
            specified in meters.";
    }
}
description
    "Bits defining granularity or semantics
    of Geo Coordinates fields.";
}
leaf location-uncertainty {
    type uint16;
    description
        "Number of centimeters of uncertainty for
        the location.";
}
leaf latitude-degrees {
    type uint8 {
        range "0 .. 90";
    }
    description
        "Latitude degrees north or south of the
        Equator (northern or southern hemisphere,
        respectively).";
}
leaf latitude-milliseconds {
    type uint24 {
        range "0 .. 3599999";
    }
    description
        "Latitude millisecond granularity (less
        than 60 minutes or a single degree).";
}
leaf longitude-degrees {
    type uint8 {
        range "0 .. 180";
    }
    description
        "Longitude degrees east or west of the
        Prime Meridian (eastern or western hemisphere,
        respectively).";
}
leaf longitude-milliseconds {
```

```

        type uint24 {
            range "0 .. 3599999";
        }
        description
            "Longitude millisecond granularity (less
            than 60 minutes or a single degree).";
    }
    leaf altitude {
        type int32;
        description
            "Height relative to sea level in
            centimeters or meters. A negative height
            indicates that the location is below sea
            level.";
    }
    leaf radius {
        type uint16;
        description
            "Radius of a circle centered at the
            specified coordinates. The radius is specified
            in meters unless the K-bit is specified
            indicating specification in kilometers. If the
            radius is specified, the geo-coordinates specify
            the entire area of the circle defined by the
            radius and center point.";
    }
}
}
}

```

<CODE ENDS>

4. IANA Routing Types YANG Module

```

<CODE BEGINS> file "iana-routing-types@2017-06-29.yang"
module iana-routing-types {
    namespace "urn:ietf:params:xml:ns:yang:iana-routing-types";
    prefix iana-rt-types;

    organization
        "IANA";
    contact
        "
            Internet Assigned Numbers Authority

            Postal: ICANN
            4676 Admiralty Way, Suite 330
            Marina del Rey, CA 90292

```

```
Tel:      +1 310 823 9358
<mailto:iana@iana.org>";
description
  "This module contains a collection of YANG data types
  considered defined by IANA and used for routing
  protocols.

  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 2017-06-29 {
  description
    "Initial revision.";
  reference "RFC TBD: IANA Routing YANG Data Types";
}

/*** Collection of IANA types related to routing ***/
/*** IANA address family Identities ***/

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

identity ipv4 {
  base address-family;
  description
    "IPv4 Address Family - IANA Registry Assigned Number: 1";
}

identity ipv6 {
  base address-family;
  description
    "IPv6 Address Family - IANA Registry Assigned Number: 2";
}
```

```
identity nsap {
  base address-family;
  description
    "OSI Network Service Access Point (NSAP) Address Family -
    IANA Registry Assigned Number: 3";
}

identity hdlc {
  base address-family;
  description
    "High-Level Data Link Control (HDLC) Address Family -
    IANA Registry Assigned Number: 4";
}

identity bbn1822 {
  base address-family;
  description
    "Bolt, Beranek, and Newman Report 1822 (BBN 1822)
    Address Family - IANA Registry Assigned Number: 5";
}

identity ieee802 {
  base address-family;
  description
    "IEEE 802 Committee Address Family (aka, MAC address) -
    IANA Registry Assigned Number: 6";
}

identity e163 {
  base address-family;
  description
    "ITU-T E.163 Address Family -
    IANA Registry Assigned Number: 7";
}

identity e164 {
  base address-family;
  description
    "ITU-T E.164 (SMDS, Frame Relay, ATM) Address Family -
    IANA Registry Assigned Number: 8";
}

identity f69 {
  base address-family;
  description
    "ITU-T F.69 (Telex) Address Family -
    IANA Registry Assigned Number: 9";
}
```

```
identity x121 {
  base address-family;
  description
    "ITU-T X.121 (X.25, Frame Relay) Address Family -
    IANA Registry Assigned Number: 10";
}

identity ipx {
  base address-family;
  description
    "Novell Internetwork Packet Exchange (IPX)
    Address Family - IANA Registry Assigned Number: 11";
}

identity appletalk {
  base address-family;
  description
    "Apple AppleTalk Address Family -
    IANA Registry Assigned Number: 12";
}

identity decnet-iv {
  base address-family;
  description
    "Digital Equipment DECnet Phase IV Address Family -
    IANA Registry Assigned Number: 13";
}

identity vines {
  base address-family;
  description
    "Banyan Vines Address Family -
    IANA Registry Assigned Number: 14";
}

identity e164-nsap {
  base address-family;
  description
    "ITU-T E.164 with NSAP sub-address Address Family -
    IANA Registry Assigned Number: 15";
}

identity dns {
  base address-family;
  description
    "Domain Name System (DNS) Address Family -
    IANA Registry Assigned Number: 16";
}
```

```
identity distinguished-name {
  base address-family;
  description
    "Distinguished Name Address Family -
     IANA Registry Assigned Number: 17";
}

identity as-num {
  base address-family;
  description
    "AS Number Family -
     IANA Registry Assigned Number: 18";
}

identity xtp-v4 {
  base address-family;
  description
    "Xpress Transport Protocol (XTP) over IPv4
     Address Family - IANA Registry Assigned Number: 19";
}

identity xtp-v6 {
  base address-family;
  description
    "Xpress Transport Protocol (XTP) over IPv4
     Address Family - IANA Registry Assigned Number: 20";
}

identity xtp-native {
  base address-family;
  description
    "Xpress Transport Protocol (XTP) native mode
     Address Family - IANA Registry Assigned Number: 21";
}

identity fc-port {
  base address-family;
  description
    "Fibre Channel (FC) World-Wide Port Name
     Address Family - IANA Registry Assigned Number: 22";
}

identity fc-node {
  base address-family;
  description
    "Fibre Channel (FC) World-Wide Node Name
     Address Family - IANA Registry Assigned Number: 23";
}
```



```
identity gwid {
  base address-family;
  description
    "ATM Gateway Identifier (GWID) Number Family -
     IANA Registry Assigned Number: 24";
}

identity l2vpn {
  base address-family;
  description
    "Layer-2 VPN (L2VPN) Address Family -
     IANA Registry Assigned Number: 25";
}

identity mpls-tp-section-eid {
  base address-family;
  description
    "MPLS-TP Section Endpoint Identifier Address Family -
     IANA Registry Assigned Number: 26";
}

identity mpls-tp-lsp-eid {
  base address-family;
  description
    "MPLS-TP LSP Endpoint Identifier Address Family -
     IANA Registry Assigned Number: 27";
}

identity mpls-tp-pwe-eid {
  base address-family;
  description
    "MPLS-TP Pseudowire Endpoint Identifier
     Address Family - IANA Registry Assigned Number: 28";
}

identity mt-v4 {
  base address-family;
  description
    "Multi-Topology IPv4 Address Family -
     Address Family - IANA Registry Assigned Number: 29";
}

identity mt-v6 {
  base address-family;
  description
    "Multi-Topology IPv6 Address Family -
     Address Family - IANA Registry Assigned Number: 30";
}
```

```
identity eigrp-common-sf {
  base address-family;
  description
    "Enhanced Interior Gateway Routing Protocol (EIGRP)
     Common Service Family Address Family -
     IANA Registry Assigned Number: 16384";
}

identity eigrp-v4-sf {
  base address-family;
  description
    "Enhanced Interior Gateway Routing Protocol (EIGRP)
     IPv4 Service Family Address Family -
     IANA Registry Assigned Number: 16385";
}

identity eigrp-v6-sf {
  base address-family;
  description
    "Enhanced Interior Gateway Routing Protocol (EIGRP)
     IPv6 Service Family Address Family -
     IANA Registry Assigned Number: 16386";
}

identity lcaf {
  base address-family;
  description
    "LISP Canonical Address Format (LCAF)
     Address Family - IANA Registry Assigned Number: 16387";
}

identity bgp-ls {
  base address-family;
  description
    "Border Gateway Protocol - Link State (BGP-LS)
     Address Family - IANA Registry Assigned Number: 16388";
}

identity mac-48 {
  base address-family;
  description
    "IEEE 48-bit Media Access Control (MAC)
     Address Family - IANA Registry Assigned Number: 16389";
}

identity mac-64 {
  base address-family;
  description
```

```
    "IEEE 64-bit Media Access Control (MAC)
    Address Family - IANA Registry Assigned Number: 16390";
}

identity trill-oui {
    base address-family;
    description
        "TRILL IEEE Organizationally Unique Identifier (OUI) -
        Address Family - IANA Registry Assigned Number: 16391";
}

identity trill-mac-24 {
    base address-family;
    description
        "TRILL Final 3 octets of 48-bit MAC address
        Address Family - IANA Registry Assigned Number: 16392";
}

identity trill-mac-48 {
    base address-family;
    description
        "TRILL Final 5 octets of 64-bit MAC address
        Address Family - IANA Registry Assigned Number: 16393";
}

identity trill-rbridge-port-id {
    base address-family;
    description
        "TRILL Remote Bridge (RBridge) Port ID
        Address Family - IANA Registry Assigned Number: 16394";
}

identity trill-nickname {
    base address-family;
    description
        "TRILL Nickname
        Address Family - IANA Registry Assigned Number: 16395";
}

/*** SAFIs for Multi-Protocol BGP Identities ***/

identity bgp-safi {
    description
        "Base identity from which identities describing BGP
        Subsequent Address Family Identifier (SAFI) - RFC 4760.";
}

identity unicast-safi {
```

```
    base bgp-safi;
    description
      "Unicast SAFI -
       IANA Registry Assigned Number: 1";
  }

  identity multicast-safi {
    base bgp-safi;
    description
      "Multicast SAFI -
       IANA Registry Assigned Number: 2";
  }

  identity labeled-unicast-safi {
    base bgp-safi;
    description
      "Labeled Unicast SAFI -
       IANA Registry Assigned Number: 4";
  }

  identity multicast-vpn-safi {
    base bgp-safi;
    description
      "Multicast VPN SAFI -
       IANA Registry Assigned Number: 5";
  }

  identity pseudowire-safi {
    base bgp-safi;
    description
      "Multi-segment Pseudowire VPN SAFI -
       IANA Registry Assigned Number: 6";
  }

  identity tunnel-enap-safi {
    base bgp-safi;
    description
      "Tunnel Encap SAFI -
       IANA Registry Assigned Number: 7";
  }

  identity mcast-vpls-safi {
    base bgp-safi;
    description
      "Multicast Virtual Private LAN Service (VPLS) SAFI -
       IANA Registry Assigned Number: 8";
  }
```

```
identity tunnel-safi {
  base bgp-safi;
  description
    "Tunnel SAFI -
     IANA Registry Assigned Number: 64";
}

identity vpls-safi {
  base bgp-safi;
  description
    "Virtual Private LAN Service (VPLS) SAFI -
     IANA Registry Assigned Number: 65";
}

identity mdt-safi {
  base bgp-safi;
  description
    "Multicast Distribution Tree (MDT) SAFI -
     IANA Registry Assigned Number: 66";
}

identity v4-over-v6-safi {
  base bgp-safi;
  description
    "IPv4 over IPv6 SAFI -
     IANA Registry Assigned Number: 67";
}

identity v6-over-v4-safi {
  base bgp-safi;
  description
    "IPv6 over IPv4 SAFI -
     IANA Registry Assigned Number: 68";
}

identity ll-vpn-auto-discovery-safi {
  base bgp-safi;
  description
    "Layer-1 VPN Auto Discovery SAFI -
     IANA Registry Assigned Number: 69";
}

identity evpn-safi {
  base bgp-safi;
  description
    "Ethernet VPN (EVPN) SAFI -
     IANA Registry Assigned Number: 70";
}
```

```
identity bgp-ls-safi {
  base bgp-safi;
  description
    "BGP Link-State (BGP-LS) SAFI -
     IANA Registry Assigned Number: 71";
}

identity bgp-ls-vpn-safi {
  base bgp-safi;
  description
    "BGP Link-State (BGP-LS) VPN SAFI -
     IANA Registry Assigned Number: 72";
}

identity sr-te-safi {
  base bgp-safi;
  description
    "Segment Routing - Traffic Engineering (SR-TE) SAFI -
     IANA Registry Assigned Number: 73";
}

identity labeled-vpn-safi {
  base bgp-safi;
  description
    "MPLS Labeled VPN SAFI -
     IANA Registry Assigned Number: 128";
}

identity multicast-mpls-vpn-safi {
  base bgp-safi;
  description
    "Multicast for BGP/MPLS IP VPN SAFI -
     IANA Registry Assigned Number: 129";
}

identity route-target-safi {
  base bgp-safi;
  description
    "Route Target SAFI -
     IANA Registry Assigned Number: 132";
}

identity ipv4-flow-spec-safi {
  base bgp-safi;
  description
    "IPv4 Flow Specification SAFI -
     IANA Registry Assigned Number: 133";
}
```

```
identity vpnv4-flow-spec-safi {  
  base bgp-safi;  
  description  
    "IPv4 VPN Flow Specification SAFI -  
    IANA Registry Assigned Number: 134";  
}
```

<CODE ENDS>

5. IANA Considerations

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

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

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

```
-----  
URI: urn:ietf:params:xml:ns:yang:iana-routing-types  
Registrant Contact: IANA  
XML: N/A, the requested URI is an XML namespace.  
-----
```

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

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

```
-----  
name:          iana-routing-types  
namespace:     urn:ietf:params:xml:ns:yang:iana-routing-types  
prefix:        iana-rt-types  
reference:     RFC XXXX  
-----
```

5.1. IANA-Maintained iana-routing-types Module

This document defines the initial version of the IANA-maintained iana-routing-types YANG module.

The iana-routing-types YANG module is intended to reflect the "Address Family Numbers" registry [IANA-ADDRESS-FAMILY-REGISTRY] and "Subsequent Address Family Identifiers (SAFI) Parameters" registry [IANA-SAFI-REGISTRY].

IANA has added this notes to the "iana-routing-types YANG Module" registry:

Address Families and Subsequent Address Families must not be directly added to the iana-routing-types YANG module. They must instead be respectively added to the "Address Family Numbers" and "Subsequent Address Family Identifiers (SAFI) Parameters" registries.

When an Address Family or Subsequent Address Family is respectively added to the "Address Family Numbers" registry or the "Subsequent Address Family Identifiers (SAFI) Parameters" registry, a new "identity" statement must be added to the iana-routing-types YANG module. The name of the "identity" is the same as the corresponding address family or SAFI only it will be a valid YANG identifier in all lowercase and with hyphens separating individual words in compound identifiers. The following substatements to the "identity" statement should be defined:

"base": Contains the value "address-family" for address families or "bgp-safi" for subsequent address families.

"status": Include only if a registration has been deprecated (use the value "deprecated") or obsoleted (use the value "obsolete").

"description": Replicate the description from the registry, if any. Insert line breaks as needed so that the line does not exceed 72 characters.

"reference": Replicate the reference from the registry, if any, and add the title of the document.

Unassigned or reserved values are not present in these modules.

When the iana-routing-types YANG module is updated, a new "revision" statement must be added in front of the existing revision statements.

IANA has added this new note to the the "Address Family Numbers" and "Subsequent Address Family Identifiers (SAFI) Parameters" registries:

When this registry is modified, the YANG module
iana-routing-types must be updated as defined in RFC XXXX.

6. Security Considerations

This document defines common data types using the YANG data modeling language. The definitions themselves have no security impact on the Internet, but the usage of these definitions in concrete YANG modules might have. The security considerations spelled out in the YANG specification [RFC7950] apply for this document as well.

7. Acknowledgements

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Ebben Aries, Lou Berger, Qin Wu, Rob Shakir, Xufeng Liu, and Yingzhen Qu.

Thanks to Martin Bjorkland, Tom Petch, Stewart Bryant, and Radek Krejci for comments on the model and document text. Thanks to Jeff Haas and Robert Raszuk for suggestions for additional common routing types.

8. References

8.1. Normative References

- [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>>.
- [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>>.

[IANA-ADDRESS-FAMILY-REGISTRY]
"IANA Address Family Registry",
<[https://www.iana.org/assignments/address-family-numbers/
address-family-numbers.xhtml#address-family-numbers-2](https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml#address-family-numbers-2)>.

[IANA-SAFI-REGISTRY]
"IANA Subsequent Address Family Identities (SAFI)
Parameters Registry", <[https://www.iana.org/assignments/
safi-namespace/safi-namespace.xhtml#safi-namespace-2](https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml#safi-namespace-2)>.

8.2. Informative References

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2008, August 2008.

[I-D.ietf-bfd-yang]
Rahman, R., Zheng, L., Networks, J., Jethanandani, M., and G. Mirsky, "Yang Data Model for Bidirectional Forwarding Detection (BFD)", draft-ietf-bfd-yang-05 (work in progress), March 2017.

[I-D.ietf-idr-bgp-model]
Shaikh, A., Shakir, R., Patel, K., Hares, S., D'Souza, K., Bansal, D., Clemm, A., Zhdankin, A., Jethanandani, M., and X. Liu, "BGP Model for Service Provider Networks", draft-ietf-idr-bgp-model-02 (work in progress), July 2016.

[I-D.ietf-ospf-yang]
Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "Yang Data Model for OSPF Protocol", draft-ietf-ospf-yang-07 (work in progress), March 2017.

[I-D.ietf-pim-yang]
Liu, X., McAllister, P., Peter, A., Sivakumar, M., Liu, Y., and f. hu, "A YANG data model for Protocol-Independent Multicast (PIM)", draft-ietf-pim-yang-08 (work in progress), April 2017.

[I-D.ietf-teas-yang-rsvp]
Beeram, V., Saad, T., Gandhi, R., Liu, X., Bryskin, I., and H. Shah, "A YANG Data Model for Resource Reservation Protocol (RSVP)", draft-ietf-teas-yang-rsvp-07 (work in progress), March 2017.

- [I-D.ietf-teas-yang-te]
Saad, T., Gandhi, R., Liu, X., Beeram, V., Shah, H., and I. Bryskin, "A YANG Data Model for Traffic Engineering Tunnels and Interfaces", draft-ietf-teas-yang-te-06 (work in progress), March 2017.
- [I-D.ietf-bess-l2vpn-yang]
Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", draft-ietf-bess-l2vpn-yang-05 (work in progress), March 2017.
- [I-D.ietf-bess-l3vpn-yang]
Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S., Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model for BGP/MPLS L3 VPNs", draft-ietf-bess-l3vpn-yang-01 (work in progress), April 2017.
- [I-D.ietf-mpls-base-yang]
Raza, K., Gandhi, R., Liu, X., Beeram, V., Saad, T., Bryskin, I., Chen, X., Jones, R., and B. Wen, "A YANG Data Model for MPLS Base", draft-ietf-mpls-base-yang-04 (work in progress), March 2017.
- [RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001, <<http://www.rfc-editor.org/info/rfc3032>>.
- [RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, DOI 10.17487/RFC3209, December 2001, <<http://www.rfc-editor.org/info/rfc3209>>.
- [RFC3471] Berger, L., Ed., "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description", RFC 3471, DOI 10.17487/RFC3471, January 2003, <<http://www.rfc-editor.org/info/rfc3471>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <<http://www.rfc-editor.org/info/rfc4364>>.
- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <<http://www.rfc-editor.org/info/rfc4664>>.

- [RFC5701] Rekhter, Y., "IPv6 Address Specific BGP Extended Community Attribute", RFC 5701, DOI 10.17487/RFC5701, November 2009, <<http://www.rfc-editor.org/info/rfc5701>>.
- [RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <<http://www.rfc-editor.org/info/rfc5880>>.
- [RFC7274] Kompella, K., Andersson, L., and A. Farrel, "Allocating and Retiring Special-Purpose MPLS Labels", RFC 7274, DOI 10.17487/RFC7274, June 2014, <<http://www.rfc-editor.org/info/rfc7274>>.
- [WGS84] National Imagery and Mapping Agency, "Department of Defense World Geodetic System 1984, Third Edition", NIMA TR83500.2, January 2000.

Authors' Addresses

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean VA 22102
USA

EMail: Xufeng_Liu@jabil.com

Yingzhen Qu
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara CA 95050
USA

EMail: yingzhen.qu@huawei.com

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

EMail: acee@cisco.com

Christian Hopps
Deutsche Telekom

EMail: chopps@chopps.org

Lou Berger
LabN Consulting, L.L.C.

EMail: lberger@labn.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2018

X. Liu
Jabil
Y. Qu
Futurewei Technologies, Inc.
A. Lindem
Cisco Systems
C. Hopps
Deutsche Telekom
L. Berger
LabN Consulting, L.L.C.
October 13, 2017

Routing Area Common YANG Data Types
draft-ietf-rtgwg-routing-types-17

Abstract

This document defines a collection of common data types using the YANG data modeling language. These derived common types are designed to be imported by other modules defined in the routing area.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 16, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	2
2. Overview	2
3. IETF Routing Types YANG Module	6
4. IANA Routing Types YANG Module	22
5. IANA Considerations	31
5.1. IANA-Maintained iana-routing-types Module	32
6. Security Considerations	33
7. Acknowledgements	33
8. References	34
8.1. Normative References	34
8.2. Informative References	34
Authors' Addresses	36

1. Introduction

The YANG [RFC6020] [RFC7950] is a data modeling language used to model configuration data, state data, Remote Procedure Calls, and notifications for network management protocols. The YANG language supports a small set of built-in data types and provides mechanisms to derive other types from the built-in types.

This document introduces a collection of common data types derived from the built-in YANG data types. The derived types are designed to be the common types applicable for modeling in the routing area.

1.1. Terminology

The terminology for describing YANG data models is found in [RFC7950].

2. Overview

This document defines the two models for common routing types, `ietf-routing-types` and `iana-routing-types`. The only module imports are from [RFC6991]. The `ietf-routing-types` model contains common routing types other than those corresponding directly to IANA mappings. These include:

`router-id`

Router Identifiers are commonly used to identify nodes in routing and other control plane protocols. An example usage of router-id can be found in [I-D.ietf-ospf-yang].

route-target

Route Targets (RTs) are commonly used to control the distribution of virtual routing and forwarding (VRF) information, see [RFC4364], in support of BGP/MPLS IP virtual private networks (VPNs) and BGP/MPLS Ethernet VPNs [RFC7432]. An example usage can be found in [I-D.ietf-bess-l2vpn-yang].

ipv6-route-target

IPv6 Route Targets (RTs) are similar to standard Route Targets only they are IPv6 Address Specific BGP Extended Communities as described in [RFC5701]. An IPv6 Route Target is 20 octets and includes an IPv6 address as the global administrator.

route-target-type

This type defines the import and export rules of Route Targets, as described in Section 4.3.1 of [RFC4364]. An example usage can be found in [I-D.ietf-idr-bgp-model].

route-distinguisher

Route Distinguishers (RDs) are commonly used to identify separate routes in support of virtual private networks (VPNs). For example, in [RFC4364], RDs are commonly used to identify independent VPNs and VRFs, and more generally, to identify multiple routes to the same prefix. An example usage can be found in [I-D.ietf-idr-bgp-model].

route-origin

Route Origin is commonly used to indicate the Site of Origin for Routing and forwarding (VRF) information, see [RFC4364], in support of BGP/MPLS IP virtual private networks (VPNs) and BGP/MPLS Ethernet VPNs [RFC7432]. An example usage can be found in [I-D.ietf-bess-l3vpn-yang].

ipv6-route-origin

An IPv6 Route Origin would also be used to indicate the Site of Origin for Routing and forwarding (VRF) information, see [RFC4364], in support of virtual private networks (VPNs). IPv6 Route Origins are IPv6 Address Specific BGP Extended Communities as described in [RFC5701]. An IPv6 Route Origin is 20 octets and includes an IPv6 address as the global administrator.

ipv4-multicast-group-address

This type defines the representation of an IPv4 multicast group address, which is in the range from 224.0.0.0 to 239.255.255.255. An example usage can be found in [I-D.ietf-pim-yang].

ipv6-multicast-group-address

This type defines the representation of an IPv6 multicast group address, which is in the range of FF00::/8. An example usage can be found in [I-D.ietf-pim-yang].

ip-multicast-group-address

This type represents an IP multicast group address and is IP version neutral. The format of the textual representation implies the IP version. An example usage can be found in [I-D.ietf-pim-yang].

ipv4-multicast-source-address

IPv4 source address type for use in multicast control protocols. This type also allows the indication of wildcard sources, i.e., "*". An example of where this type may/will be used is [I-D.ietf-pim-yang].

ipv6-multicast-source-address

IPv6 source address type for use in multicast control protocols. This type also allows the indication of wildcard sources, i.e., "*". An example of where this type may/will be used is [I-D.ietf-pim-yang].

bandwidth-ieee-float32

Bandwidth in IEEE 754 floating point 32-bit binary format [IEEE754]. Commonly used in Traffic Engineering control plane protocols. An example of where this type may/will be used is [I-D.ietf-ospf-yang].

link-access-type

This type identifies the IGP link type. An example of where this type may/will be used is [I-D.ietf-ospf-yang].

timer-multiplier

This type is used in conjunction with a timer-value type. It is generally used to indicate define the number of timer-value intervals that may expire before a specific event must occur. Examples of this include the arrival of any BFD packets, see [RFC5880] Section 6.8.4, or hello_interval in [RFC3209]. Example of where this type may/will be used is [I-D.ietf-idr-bgp-model] and [I-D.ietf-teas-yang-rsvp].

timer-value-seconds16

This type covers timers which can be set in seconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint16 (2 octets). An example of where this type may/will be used is [I-D.ietf-ospf-yang].

timer-value-seconds32

This type covers timers which can be set in seconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint32 (4 octets). An example of where this type may/will be used is [I-D.ietf-teas-yang-rsvp].

timer-value-milliseconds

This type covers timers which can be set in milliseconds, not set, or set to infinity. This type supports a range of values that can be represented in a uint32 (4 octets). Examples of where this type may/will be used include [I-D.ietf-teas-yang-rsvp] and [I-D.ietf-bfd-yang].

percentage

This type defines a percentage with a range of 0-100%. An example usage can be found in [I-D.ietf-idr-bgp-model].

timeticks64

This type is based on the timeticks type defined in [RFC6991] but with 64-bit precision. It represents the time in hundredths of a second between two epochs. An example usage can be found in [I-D.ietf-idr-bgp-model].

uint24

This type defines a 24-bit unsigned integer. It is used by [I-D.ietf-ospf-yang].

generalized-label

This type represents a generalized label for Generalized Multi-Protocol Label Switching (GMPLS) [RFC3471]. The Generalized Label does not identify its type, which is known from the context. An example usage can be found in [I-D.ietf-teas-yang-tel].

mpls-label-special-purpose

This type represents the special-purpose Multiprotocol Label Switching (MPLS) label values [RFC7274]. An example usage can be found in [I-D.ietf-mpls-base-yang].

mpls-label-general-use

The 20 bits label values in an MPLS label stack entry, specified in [RFC3032]. This label value does not include the encodings of Traffic Class and TTL (time to live). The label range specified by this type is for general use, with special-purpose MPLS label

values excluded. An example usage can be found in [I-D.ietf-mpls-base-yang].

mpls-label

The 20 bits label values in an MPLS label stack entry, specified in [RFC3032]. This label value does not include the encodings of Traffic Class and TTL (time to live). The label range specified by this type covers the general use values and the special-purpose label values. An example usage can be found in [I-D.ietf-mpls-base-yang].

This document defines the following YANG groupings:

mpls-label-stack

This grouping defines a reusable collection of schema nodes representing an MPLS label stack [RFC3032]. An example usage can be found in [I-D.ietf-mpls-base-yang].

vpn-route-targets

This grouping defines a reusable collection of schema nodes representing Route Target import-export rules used in the BGP enabled Virtual Private Networks (VPNs). [RFC4364][RFC4664]. An example usage can be found in [I-D.ietf-bess-l2vpn-yang].

The iana-routing-types model contains common routing types corresponding directly to IANA mappings. These include:

address-family

This type defines values for use in address family identifiers. The values are based on the IANA Address Family Numbers Registry [IANA-ADDRESS-FAMILY-REGISTRY]. An example usage can be found in [I-D.ietf-idr-bgp-model].

subsequent-address-family

This type defines values for use in subsequent address family (SAFI) identifiers. The values are based on the IANA Subsequent Address Family Identifiers (SAFI) Parameters Registry [IANA-SAFI-REGISTRY].

3. IETF Routing Types YANG Module

```
<CODE BEGINS> file "ietf-routing-types@2017-10-13.yang"
module ietf-routing-types {
  namespace "urn:ietf:params:xml:ns:yang:ietf-routing-types";
  prefix rt-types;

  import ietf-yang-types {
    prefix yang;
```

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

organization
  "IETF RTGWG - Routing Area Working Group";
contact
  "WG Web:    <http://tools.ietf.org/wg/rtgwg/>
  WG List:    <mailto:rtgwg@ietf.org>

  Editor:     Xufeng Liu
               <mailto:Xufeng_Liu@jabail.com>
               Yingzhen Qu
               <mailto:yingzhen.qu@huawei.com>
               Acee Lindem
               <mailto:acee@cisco.com>
               Christian Hopps
               <mailto:chopps@chopps.org>
               Lou Berger
               <mailto:lberger@labn.com>";
description
  "This module contains a collection of YANG data types
  considered generally useful for routing protocols.

  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 2017-10-13 {
  description "Initial revision.";
  reference "RFC TBD: Routing YANG Data Types";
}

/*** Identities related to MPLS/GMPLS ***/

identity mpls-label-special-purpose-value {
  description
```

```
    "Base identity for deriving identities describing
    special-purpose Multiprotocol Label Switching (MPLS) label
    values.";
  reference
    "RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels.";
}

identity ipv4-explicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the IPv4 Explicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity router-alert-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Router Alert Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity ipv6-explicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the IPv6 Explicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity implicit-null-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Implicit NULL Label.";
  reference "RFC3032: MPLS Label Stack Encoding. Section 2.1.";
}

identity entropy-label-indicator {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Entropy Label Indicator.";
  reference
    "RFC6790: The Use of Entropy Labels in MPLS Forwarding.
    Sections 3 and 10.1.";
}

identity gal-label {
  base mpls-label-special-purpose-value;
  description
```

```
    "This identity represents the Generic Associated Channel
    Label (GAL).";
  reference
    "RFC5586: MPLS Generic Associated Channel.
    Sections 4 and 10.";
}

identity oam-alert-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the OAM Alert Label.";
  reference
    "RFC3429: Assignment of the 'OAM Alert Label' for
    Multiprotocol Label Switching Architecture (MPLS)
    Operation and Maintenance (OAM) Functions.
    Sections 3 and 6.";
}

identity extension-label {
  base mpls-label-special-purpose-value;
  description
    "This identity represents the Extension Label.";
  reference
    "RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels. Sections 3.1 and 5.";
}

/*** Collection of types related to routing ***/

typedef router-id {
  type yang:dotted-quad;
  description
    "A 32-bit number in the dotted quad format assigned to each
    router. This number uniquely identifies the router within
    an Autonomous System.";
}

/*** Collection of types related to VPN ***/

typedef route-target {
  type string {
    pattern
      '(0:(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2})|'
      + '6[0-4][0-9]{3})|'
      + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0):(429496729[0-5]|'
      + '42949672[0-8][0-9]|'
      + '4294967[01][0-9]{2}|429496[0-6][0-9]{3})|'
      + '42949[0-5][0-9]{4})|'
```

```

+      '4294[0-8][0-9]{5}|429[0-3][0-9]{6}|'
+      '42[0-8][0-9]{7}|4[01][0-9]{8}|'
+      '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0))|'
+      '(1:(((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|'
+      '25[0-5])\.){3}([0-9]|[1-9][0-9]|'
+      '1[0-9]{2}|2[0-4][0-9]|25[0-5])):(6553[0-5]|'
+      '655[0-2][0-9]|'
+      '65[0-4][0-9]{2}|6[0-4][0-9]{3}|'
+      '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
+      '(2:(429496729[0-5]|42949672[0-8][0-9]|'
+      '4294967[01][0-9]{2}|'
+      '429496[0-6][0-9]{3}|42949[0-5][0-9]{4}|'
+      '4294[0-8][0-9]{5}|'
+      '429[0-3][0-9]{6}|42[0-8][0-9]{7}|4[01][0-9]{8}|'
+      '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0):'
+      '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+      '6[0-4][0-9]{3}|'
+      '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
+      '(6:[a-fA-F0-9]{2}){6})|'
+      '(((3-57-9a-fA-F)|[1-9a-fA-F][0-9a-fA-F]{1,3}):'
+      '[0-9a-fA-F]{1,12}))';
}

```

description

"A route target is an 8-octet BGP extended community initially identifying a set of sites in a BGP VPN (RFC 4364). However, it has since taken on a more general role in BGP route filtering.

A route target consists of two or three fields: a 2-octet type field, an administrator field, and, optionally, an assigned number field.

According to the data formats for type 0, 1, 2, and 6 defined in RFC4360, RFC5668, and RFC7432, the encoding pattern is defined as:

```

0:2-octet-asn:4-octet-number
1:4-octet-ipv4addr:2-octet-number
2:4-octet-asn:2-octet-number.
6:6-octet-mac-address.

```

Additionally, a generic pattern is defined for future route target types:

```
2-octet-other-hex-number:6-octet-hex-number
```

Some valid examples are: 0:100:100, 1:1.1.1.1:100, 2:1234567890:203 and 6:26:00:08:92:78:00";

reference

```

    "RFC4360: BGP Extended Communities Attribute.
    RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs)
    RFC5668: 4-Octet AS Specific BGP Extended Community.
    RFC7432: BGP MPLS-Based Ethernet VPN";
}

typedef ipv6-route-target {
  type string {
    pattern
      '(:|0-9a-fA-F){0,4}:)(0-9a-fA-F){0,4}:){0,5}'
      + '(((0-9a-fA-F){0,4}:)?(:|0-9a-fA-F){0,4}))|'
      + '(((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}'
      + '(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])))'
      + ':'
      + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
      + '6[0-4][0-9]{3}|'
      + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
    pattern '(((^[^:]+:){6}([^[^:]+:[^:]+)|(.*\..*)))|'
      + '(((^[^:]+:)*[^[^:]+)?::([^[^:]+)*[^[^:]+]?))'
      + ':'
      + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
      + '6[0-4][0-9]{3}|'
      + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
  }
  description
    "An IPv6 route target is a 20-octet BGP IPv6 address
    specific extended community serving the same function
    as a standard 8-octet route target only allowing for
    an IPv6 address as the global administrator. The format
    is <ipv6-address:2-octet-number>.

    Some valid examples are: 2001:DB8::1:6544 and
    2001:DB8::5eb1:791:6b37:17958";
  reference
    "RFC5701: IPv6 Address Specific BGP Extended Community
    Attribute";
}

typedef route-target-type {
  type enumeration {
    enum "import" {
      value 0;
      description
        "The route target applies to route import.";
    }
    enum "export" {
      value 1;
      description

```



```

        "The route target applies to route export.";
    }
    enum "both" {
        value 2;
        description
            "The route target applies to both route import and
            route export.";
    }
}
description
    "Indicates the role a route target takes
    in route filtering.";
reference "RFC4364: BGP/MPLS IP Virtual Private Networks
(VPNs).";
}

typedef route-distinguisher {
    type string {
        pattern
            '(0:(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2})|'
        + '6[0-4][0-9]{3})|'
        + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0):(429496729[0-5]|'
        + '42949672[0-8][0-9]|'
        + '4294967[01][0-9]{2}|429496[0-6][0-9]{3})|'
        + '42949[0-5][0-9]{4})|'
        + '4294[0-8][0-9]{5}|429[0-3][0-9]{6})|'
        + '42[0-8][0-9]{7}|4[01][0-9]{8})|'
        + '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0))|'
        + '(1:(((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9])|'
        + '25[0-5])\.){3}([0-9]|1[0-9][0-9]|'
        + '1[0-9]{2}|2[0-4][0-9]|25[0-5])):(6553[0-5]|'
        + '655[0-2][0-9]|'
        + '65[0-4][0-9]{2}|6[0-4][0-9]{3})|'
        + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
        + '(2:(429496729[0-5]|42949672[0-8][0-9]|'
        + '4294967[01][0-9]{2}|'
        + '429496[0-6][0-9]{3}|42949[0-5][0-9]{4})|'
        + '4294[0-8][0-9]{5}|'
        + '429[0-3][0-9]{6}|42[0-8][0-9]{7}|4[01][0-9]{8})|'
        + '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0):'
        + '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2})|'
        + '6[0-4][0-9]{3})|'
        + '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
        + '(6(:[a-fA-F0-9]{2}){6})|'
        + '([3-57-9a-fA-F]|1-9a-fA-F)[0-9a-fA-F]{1,3}):'
        + '[0-9a-fA-F]{1,12})';
    }
}
description

```

"A route distinguisher is an 8-octet value used to distinguish routes from different BGP VPNs (RFC 4364). As per RFC 4360, a route distinguisher will have the same format as a route target and will consist of two or three fields including a 2-octet type field, an administrator field, and, optionally, an assigned number field.

According to the data formats for type 0, 1, 2, and 6 defined in RFC4360, RFC5668, and RFC7432, the encoding pattern is defined as:

```
0:2-octet-asn:4-octet-number
1:4-octet-ipv4addr:2-octet-number
2:4-octet-asn:2-octet-number.
6:6-octet-mac-address.
```

Additionally, a generic pattern is defined for future route discriminator types:

```
2-octet-other-hex-number:6-octet-hex-number
```

Some valid examples are: 0:100:100, 1:1.1.1.1:100, 2:1234567890:203 and 6:26:00:08:92:78:00";

reference

```
"RFC4360: BGP Extended Communities Attribute.
RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs)
RFC5668: 4-Octet AS Specific BGP Extended Community.
RFC7432: BGP MPLS-Based Ethernet VPN";
```

}

```
typedef route-origin {
  type string {
    pattern
      '(0:(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]){2}|'
    +   '6[0-4][0-9]{3}|'
    +   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0):(429496729[0-5]|'
    +   '42949672[0-8][0-9]|'
    +   '4294967[01][0-9]{2}|429496[0-6][0-9]{3}|'
    +   '42949[0-5][0-9]{4}|'
    +   '4294[0-8][0-9]{5}|429[0-3][0-9]{6}|'
    +   '42[0-8][0-9]{7}|4[01][0-9]{8}|'
    +   '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0))|'
    +   '(1:(((0-9)|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|'
    +   '25[0-5])\.)}{3}([0-9]|[1-9][0-9]|'
    +   '1[0-9]{2}|2[0-4][0-9]|25[0-5])):(6553[0-5]|'
    +   '655[0-2][0-9]|'
    +   '65[0-4][0-9]{2}|6[0-4][0-9]{3}|'
    +   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
```

```

+ '(2:(429496729[0-5]|42949672[0-8][0-9]|'
+   '4294967[01][0-9]{2}|'
+   '429496[0-6][0-9]{3}|42949[0-5][0-9]{4}|'
+   '4294[0-8][0-9]{5}|'
+   '429[0-3][0-9]{6}|42[0-8][0-9]{7}|4[01][0-9]{8}|'
+   '[1-3][0-9]{9}|[1-9][0-9]{0,8}|0):'
+   '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+   '6[0-4][0-9]{3}|'
+   '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0))|'
+ '(6(:[a-fA-F0-9]{2}){6})|'
+ '(((3-57-9a-fA-F)|[1-9a-fA-F][0-9a-fA-F]{1,3}):'
+   '[0-9a-fA-F]{1,12}))';

```

```

}
description

```

"A route origin is an 8-octet BGP extended community identifying the set of sites where the BGP route originated (RFC 4364). A route target consists of two or three fields: a 2-octet type field, an administrator field, and, optionally, an assigned number field.

According to the data formats for type 0, 1, 2, and 6 defined in RFC4360, RFC5668, and RFC7432, the encoding pattern is defined as:

```

0:2-octet-asn:4-octet-number
1:4-octet-ipv4addr:2-octet-number
2:4-octet-asn:2-octet-number.
6:6-octet-mac-address.

```

Additionally, a generic pattern is defined for future route origin types:

```

2-octet-other-hex-number:6-octet-hex-number

```

Some valid examples are: 0:100:100, 1:1.1.1.1:100, 2:1234567890:203 and 6:26:00:08:92:78:00";

```

reference

```

```

"RFC4360: BGP Extended Communities Attribute.
RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs)
RFC5668: 4-Octet AS Specific BGP Extended Community.
RFC7432: BGP MPLS-Based Ethernet VPN";

```

```

}

```

```

typedef ipv6-route-origin {
  type string {
    pattern
      '(((:[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}'
      + '((([0-9a-fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|'

```

```

+ '((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.)\{3\}'
+ '(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9]))'
+ ':'
+ '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+ '6[0-4][0-9]{3}|'
+ '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
pattern '(((^[^:]+:){6}([^[^:]+:[^:]+)|(.*\..*)))|'
+ '(((^[^:]+:)*[^[^:]+)?::([^[^:]+)*[^[^:]+]?))'
+ ':'
+ '(6553[0-5]|655[0-2][0-9]|65[0-4][0-9]{2}|'
+ '6[0-4][0-9]{3}|'
+ '[1-5][0-9]{4}|[1-9][0-9]{0,3}|0)';
}
description
  "An IPv6 route origin is a 20-octet BGP IPv6 address
  specific extended community serving the same function
  as a standard 8-octet route only allowing for
  an IPv6 address as the global administrator. The format
  is <ipv6-address:2-octet-number>.

  Some valid examples are: 2001:DB8::1:6544 and
  2001:DB8::5eb1:791:6b37:17958";
reference
  "RFC5701: IPv6 Address Specific BGP Extended Community
  Attribute";
}

/*** Collection of types common to multicast ***/

typedef ipv4-multicast-group-address {
  type inet:ipv4-address {
    pattern '(2((2[4-9])|(3[0-9]))\.)\.*';
  }
  description
    "This type represents an IPv4 multicast group address,
    which is in the range from 224.0.0.0 to 239.255.255.255.";
  reference "RFC1112: Host Extensions for IP Multicasting.";
}

typedef ipv6-multicast-group-address {
  type inet:ipv6-address {
    pattern '([fF]{2}[0-9a-fA-F]{2})\.*';
  }
  description
    "This type represents an IPv6 multicast group address,
    which is in the range of FF00::/8.";
  reference

```

```
    "RFC4291: IP Version 6 Addressing Architecture. Sec 2.7.
    RFC7346: IPv6 Multicast Address Scopes.";
}

typedef ip-multicast-group-address {
    type union {
        type ipv4-multicast-group-address;
        type ipv6-multicast-group-address;
    }
    description
        "This type represents a version-neutral IP multicast group
        address. The format of the textual representation implies
        the IP version.";
}

typedef ipv4-multicast-source-address {
    type union {
        type enumeration {
            enum "*" {
                description
                    "Any source address.";
            }
        }
        type inet:ipv4-address;
    }
    description
        "Multicast source IPv4 address type.";
}

typedef ipv6-multicast-source-address {
    type union {
        type enumeration {
            enum "*" {
                description
                    "Any source address.";
            }
        }
        type inet:ipv6-address;
    }
    description
        "Multicast source IPv6 address type.";
}

/** Collection of types common to protocols */

typedef bandwidth-ieee-float32 {
    type string {
        pattern
```

```

    '0[xX](0((\.0?)?[pP](\+)?0?|(\.0?))|'
+ '1(\.([0-9a-fA-F]{0,5}[02468aAcCeE]?))?[pP](\+)?(12[0-7]|'
+ '1[01][0-9]|0?[0-9]?[0-9]))';
}
description
  "Bandwidth in IEEE 754 floating point 32-bit binary format:
   $(-1)^{(S)} * 2^{(Exponent-127)} * (1 + Fraction)$ ,
  where Exponent uses 8 bits, and Fraction uses 23 bits.
  The units are octets per second.
  The encoding format is the external hexadecimal-significant
  character sequences specified in IEEE 754 and C99. The
  format is restricted to be normalized, non-negative, and
  non-fraction: 0x1.hhhhhhp{+}d, 0X1.HHHHHHP{+}D, or 0x0p0,
  where 'h' and 'H' are hexadecimal digits and 'd' and 'D' are
  integers in the range of [0..127].
  When six hexadecimal digits are used for 'hhhhh' or
  'HHHHH', the least significant digit must be an even
  number. 'x' and 'X' indicate hexadecimal; 'p' and 'P'
  indicate power of two. Some examples are: 0x0p0, 0x1p10, and
  0x1.abcde2p+20";
reference
  "IEEE Std 754-2008: IEEE Standard for Floating-Point
  Arithmetic.";
}

typedef link-access-type {
  type enumeration {
    enum "broadcast" {
      description
        "Specify broadcast multi-access network.";
    }
    enum "non-broadcast-multiaccess" {
      description
        "Specify Non-Broadcast Multi-Access (NBMA) network.";
    }
    enum "point-to-multipoint" {
      description
        "Specify point-to-multipoint network.";
    }
    enum "point-to-point" {
      description
        "Specify point-to-point network.";
    }
  }
  description
    "Link access type.";
}

```

```
typedef timer-multiplier {
  type uint8;
  description
    "The number of timer value intervals that should be
    interpreted as a failure.";
}

typedef timer-value-seconds16 {
  type union {
    type uint16 {
      range "1..65535";
    }
    type enumeration {
      enum "infinity" {
        description
          "The timer is set to infinity.";
      }
      enum "not-set" {
        description
          "The timer is not set.";
      }
    }
  }
  units "seconds";
  description
    "Timer value type, in seconds (16-bit range).";
}

typedef timer-value-seconds32 {
  type union {
    type uint32 {
      range "1..4294967295";
    }
    type enumeration {
      enum "infinity" {
        description
          "The timer is set to infinity.";
      }
      enum "not-set" {
        description
          "The timer is not set.";
      }
    }
  }
  units "seconds";
  description
    "Timer value type, in seconds (32-bit range).";
}
```

```
typedef timer-value-milliseconds {
  type union {
    type uint32 {
      range "1..4294967295";
    }
    type enumeration {
      enum "infinity" {
        description
          "The timer is set to infinity.";
      }
      enum "not-set" {
        description
          "The timer is not set.";
      }
    }
  }
  units "milliseconds";
  description
    "Timer value type, in milliseconds.";
}

typedef percentage {
  type uint8 {
    range "0..100";
  }
  description
    "Integer indicating a percentage value";
}

typedef timeticks64 {
  type uint64;
  description
    "This type is based on the timeticks type defined in
    RFC 6991, but with 64-bit width. It represents the time,
    modulo 2^64, in hundredths of a second between two epochs.";
  reference "RFC 6991 - Common YANG Data Types";
}

typedef uint24 {
  type uint32 {
    range "0 .. 16777215";
  }
  description
    "24-bit unsigned integer";
}

/*** Collection of types related to MPLS/GMPLS ***/
```



```
typedef generalized-label {
  type binary;
  description
    "Generalized label. Nodes sending and receiving the
    Generalized Label are aware of the link-specific
    label context and type.";
  reference "RFC3471: Section 3.2";
}

typedef mpls-label-special-purpose {
  type identityref {
    base mpls-label-special-purpose-value;
  }
  description
    "This type represents the special-purpose Multiprotocol Label
    Switching (MPLS) label values.";
  reference
    "RFC3032: MPLS Label Stack Encoding.
    RFC7274: Allocating and Retiring Special-Purpose MPLS
    Labels.";
}

typedef mpls-label-general-use {
  type uint32 {
    range "16..1048575";
  }
  description
    "The 20-bit label values in an MPLS label stack entry,
    specified in RFC3032. This label value does not include
    the encodings of Traffic Class and TTL (time to live).
    The label range specified by this type is for general use,
    with special-purpose MPLS label values excluded.";
  reference "RFC3032: MPLS Label Stack Encoding.";
}

typedef mpls-label {
  type union {
    type mpls-label-special-purpose;
    type mpls-label-general-use;
  }
  description
    "The 20-bit label values in an MPLS label stack entry,
    specified in RFC3032. This label value does not include
    the encodings of Traffic Class and TTL (time to live).";
  reference "RFC3032: MPLS Label Stack Encoding.";
}

/*** Groupings **/
```

```
grouping mpls-label-stack {
  description
    "This grouping specifies an MPLS label stack. The label
    stack is encoded as a list of label stack entries. The
    list key is an identifier which indicates relative
    ordering of each entry, with the lowest value identifier
    corresponding to the top of the label stack.";
  container mpls-label-stack {
    description
      "Container for a list of MPLS label stack entries.";
    list entry {
      key "id";
      description
        "List of MPLS label stack entries.";
      leaf id {
        type uint8;
        description
          "Identifies the entry in a sequence of MPLS label
          stack entries. An entry with a smaller identifier
          value precedes an entry with a larger identifier
          value in the label stack. The value of this ID has
          no semantic meaning other than relative ordering
          and referencing the entry.";
      }
      leaf label {
        type rt-types:mpls-label;
        description
          "Label value.";
      }
      leaf ttl {
        type uint8;
        description
          "Time to Live (TTL).";
        reference "RFC3032: MPLS Label Stack Encoding.";
      }
      leaf traffic-class {
        type uint8 {
          range "0..7";
        }
        description
          "Traffic Class (TC).";
        reference
          "RFC5462: Multiprotocol Label Switching (MPLS) Label
          Stack Entry: 'EXP' Field Renamed to 'Traffic Class'
          Field.";
      }
    }
  }
}
```

```

    }

    grouping vpn-route-targets {
      description
        "A grouping that specifies Route Target import-export rules
        used in the BGP enabled Virtual Private Networks (VPNs).";
      reference
        "RFC4364: BGP/MPLS IP Virtual Private Networks (VPNs).
        RFC4664: Framework for Layer 2 Virtual Private Networks
        (L2VPNs)";
      list vpn-target {
        key "route-target";
        description
          "List of Route Targets.";
        leaf route-target {
          type rt-types:route-target;
          description
            "Route Target value";
        }
        leaf route-target-type {
          type rt-types:route-target-type;
          mandatory true;
          description
            "Import/export type of the Route Target.";
        }
      }
    }
  }
}

```

<CODE ENDS>

4. IANA Routing Types YANG Module

```

<CODE BEGINS> file "iana-routing-types@2017-09-19.yang"
module iana-routing-types {
  namespace "urn:ietf:params:xml:ns:yang:iana-routing-types";
  prefix iana-rt-types;

  organization
    "IANA";
  contact
    "
      Internet Assigned Numbers Authority

      Postal: ICANN
      4676 Admiralty Way, Suite 330
      Marina del Rey, CA 90292

      Tel: +1 310 823 9358
    "

```

```
<mailto:iana@iana.org>";
description
  "This module contains a collection of YANG data types
   considered defined by IANA and used for routing
   protocols.

  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 2017-09-19 {
  description "Initial revision.";
  reference "RFC TBD: IANA Routing YANG Data Types";
}

/** Collection of IANA types related to routing */
/** IANA address family enumeration */

typedef address-family {
  type enumeration {
    enum ipv4 {
      value 1;
      description "IPv4 Address Family";
    }

    enum ipv6 {
      value 2;
      description "IPv6 Address Family";
    }

    enum nsap {
      value 3;
      description "OSI Network Service Access Point (NSAP)
                  Address Family";
    }

    enum hdlc {
      value 4;
    }
  }
}
```

```
        description "High-Level Data Link Control (HDLC)
                    Address Family";
    }

    enum bbn1822 {
        value 5;
        description "Bolt, Beranek, and Newman Report
                    1822 (BBN 1822) Address Family";
    }

    enum ieee802 {
        value 6;
        description "IEEE 802 Committee Address Family (aka,
                    MAC address)";
    }

    enum e163 {
        value 7;
        description "ITU-T E.163 Address Family";
    }

    enum e164 {
        value 8;
        description "ITU-T E.164 (SMDS, Frame Relay, ATM)
                    Address Family";
    }

    enum f69 {
        value 9;
        description "ITU-T F.69 (Telex) Address Family";
    }

    enum x121 {
        value 10;
        description "ITU-T X.121 (X.25, Frame Relay)
                    Address Family";
    }

    enum ipx {
        value 11;
        description "Novell Internetwork Packet Exchange (IPX)
                    Address Family";
    }

    enum appletalk {
        value 12;
        description "Apple AppleTalk Address Family";
    }
```

```
    }

    enum decnet-iv {
        value 13;
        description "Digital Equipment DECnet Phase IV
                    Address Family";
    }

    enum vines {
        value 14;
        description "Banyan Vines Address Family";
    }

    enum el64-nsap {
        value 15;
        description "ITU-T E.164 with NSAP sub-address
                    Address Family";
    }

    enum dns {
        value 16;
        description "Domain Name System (DNS) Address
                    Family";
    }

    enum distinguished-name {
        value 17;
        description "Distinguished Name Address Family";
    }

    enum as-num {
        value 18;
        description "AS Number Address Family";
    }

    enum xtp-v4 {
        value 19;
        description "Xpress Transport Protocol (XTP) over IPv4
                    Address Family";
    }

    enum xtp-v6 {
        value 20;
        description "Xpress Transport Protocol (XTP) over IPv6
                    Address Family";
    }

    enum xtp-native {
```

```
    value 21;
    description "Xpress Transport Protocol (XTP) native mode
                Address Family";
}

enum fc-port {
    value 22;
    description "Fibre Channel (FC) World-Wide Port Name
                Address Family";
}

enum fc-node {
    value 23;
    description "Fibre Channel (FC) World-Wide Node Name
                Address Family";
}

enum gwid {
    value 24;
    description "ATM Gateway Identifier (GWID) Number Address Family";
}

enum l2vpn {
    value 25;
    description "Layer-2 VPN (L2VPN) Address Family";
}

enum mpls-tp-section-eid {
    value 26;
    description "MPLS-TP Section Endpoint Identifier
                Address Family";
}

enum mpls-tp-lsp-eid {
    value 27;
    description "MPLS-TP LSP Endpoint Identifier
                Address Family";
}

enum mpls-tp-pwe-eid {
    value 28;
    description "MPLS-TP Pseudowire Endpoint Identifier
                Address Family";
}

enum mt-v4 {
    value 29;
```

```
    description "Multi-Topology IPv4 Address Family";
  }

  enum mt-v6 {
    value 30;
    description "Multi-Topology IPv6 Address Family";
  }

  enum eigrp-common-sf {
    value 16384;
    description "Enhanced Interior Gateway Routing Protocol
                 (EIGRP) Common Service Family Address
                 Family";
  }

  enum eigrp-v4-sf {
    value 16385;
    description "Enhanced Interior Gateway Routing Protocol
                 (EIGRP) IPv4 Service Family Address Family";
  }

  enum eigrp-v6-sf {
    value 16386;
    description "Enhanced Interior Gateway Routing Protocol
                 (EIGRP) IPv6 Service Family Address Family";
  }

  enum lcaf {
    value 16387;
    description "LISP Canonical Address Format (LCAF)
                 Address Family";
  }

  enum bgp-ls {
    value 16388;
    description "Border Gateway Protocol - Link State (BGP-LS)
                 Address Family";
  }

  enum mac-48 {
    value 16389;
    description "IEEE 48-bit Media Access Control (MAC)
                 Address Family";
  }

  enum mac-64 {
    value 16390;
    description "IEEE 64-bit Media Access Control (MAC)";
  }
```



```
        Address Family";
    }

    enum trill-oui {
        value 16391;
        description "TRILL IEEE Organizationally Unique
                    Identifier (OUI) Address Family";
    }

    enum trill-mac-24 {
        value 16392;
        description "TRILL Final 3 octets of 48-bit MAC
                    address Address Family";
    }

    enum trill-mac-40 {
        value 16393;
        description "TRILL Final 5 octets of 64-bit MAC
                    address Address Family";
    }

    enum ipv6-64 {
        value 16394;
        description "First 8 octets (64-bits) of an IPv6
                    address Address Family";
    }

    enum trill-rbridge-port-id {
        value 16395;
        description "TRILL Remote Bridge (RBridge) Port ID
                    Address Family";
    }

    enum trill-nickname {
        value 16396;
        description "TRILL Nickname Address Family";
    }
}
description "Enumeration containing all the IANA
            defined address families.";

}

/**** SAFIs for Multi-Protocol BGP enumeration ****/

typedef bgp-safi {
    type enumeration {
        enum unicast-safi {
```

```
    value 1;
    description "Unicast SAFI";
}

enum multicast-safi {
    value 2;
    description "Multicast SAFI";
}

enum labeled-unicast-safi {
    value 4;
    description "Labeled Unicast SAFI";
}

enum multicast-vpn-safi {
    value 5;
    description "Multicast VPN SAFI";
}

enum pseudowire-safi {
    value 6;
    description "Multi-segment Pseudowire VPN SAFI";
}

enum tunnel-encap-safi {
    value 7;
    description "Tunnel Encap SAFI";
}

enum mcast-vpls-safi {
    value 8;
    description "Multicast Virtual Private LAN Service
                  (VPLS) SAFI";
}

enum tunnel-safi {
    value 64;
    description "Tunnel SAFI";
}

enum vpls-safi {
    value 65;
    description "Virtual Private LAN Service (VPLS) SAFI";
}

enum mdt-safi {
    value 66;
    description "Multicast Distribution Tree (MDT) SAFI";
}
```

```
    }

    enum v4-over-v6-safi {
        value 67;
        description "IPv4 over IPv6 SAFI";
    }

    enum v6-over-v4-safi {
        value 68;
        description "IPv6 over IPv4 SAFI";
    }

    enum l1-vpn-auto-discovery-safi {
        value 69;
        description "Layer-1 VPN Auto Discovery SAFI";
    }

    enum evpn-safi {
        value 70;
        description "Ethernet VPN (EVPN) SAFI";
    }

    enum bgp-ls-safi {
        value 71;
        description "BGP Link-State (BGP-LS) SAFI";
    }

    enum bgp-ls-vpn-safi {
        value 72;
        description "BGP Link-State (BGP-LS) VPN SAFI";
    }

    enum sr-te-safi {
        value 73;
        description "Segment Routing - Traffic Engineering
                    (SR-TE) SAFI";
    }

    enum labeled-vpn-safi {
        value 128;
        description "MPLS Labeled VPN SAFI";
    }

    enum multicast-mpls-vpn-safi {
        value 129;
        description "Multicast for BGP/MPLS IP VPN SAFI";
    }
```

```
enum route-target-safi {
  value 132;
  description "Route Target SAFI";
}

enum ipv4-flow-spec-safi {
  value 133;
  description "IPv4 Flow Specification SAFI";
}

enum vpnv4-flow-spec-safi {
  value 134;
  description "IPv4 VPN Flow Specification SAFI";
}

enum vpn-auto-discovery-safi {
  value 140;
  description "VPN Auto-Discovery SAFI";
}
}
description "Enumeration for BGP Subsequent Address
             Family Identifier (SAFI) - RFC 4760."
}
}

<CODE ENDS>
```

5. IANA Considerations

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

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

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

```
-----
URI: urn:ietf:params:xml:ns:yang:iana-routing-types
Registrant Contact: IANA
XML: N/A, the requested URI is an XML namespace.
-----
```

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

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

```
-----
name:      iana-routing-types
namespace: urn:ietf:params:xml:ns:yang:iana-routing-types
prefix:    iana-rt-types
reference:  RFC XXXX
-----
```

5.1. IANA-Maintained iana-routing-types Module

This document defines the initial version of the IANA-maintained iana-routing-types YANG module Section 4.

The iana-routing-types YANG module is intended to reflect the "Address Family Numbers" registry [IANA-ADDRESS-FAMILY-REGISTRY] and "Subsequent Address Family Identifiers (SAFI) Parameters" registry [IANA-SAFI-REGISTRY].

IANA has added this notes to the "iana-routing-types YANG Module" registry:

Address Families and Subsequent Address Families must not be directly added to the iana-routing-types YANG module. They must instead be respectively added to the "Address Family Numbers" and "Subsequent Address Family Identifiers (SAFI) Parameters" registries.

When an Address Family or Subsequent Address Family is respectively added to the "Address Family Numbers" registry or the "Subsequent Address Family Identifiers (SAFI) Parameters" registry, a new "enum" statement must be added to the iana-routing-types YANG module. The name of the "enum" is the same as the corresponding address family or SAFI only it will be a valid YANG identifier in all lowercase and with hyphens separating individual words in compound identifiers. The following substatements to the "enum" statement should be defined:

"enum": Contains the YANG enum identifier for the address-family or "bgp-safi" for subsequent address families. This may be the same as the address-family or "bgp-safi" or it may be a shorter version to facilitate YANG identifier usage.

"value": Contains the IANA assigned value corresponding to the address-family or "bgp-safi" for subsequent address families.

"status": Include only if a registration has been deprecated (use the value "deprecated") or obsoleted (use the value "obsolete").

"description": Replicate the description from the registry, if any. Insert line breaks as needed so that the line does not exceed 72 characters.

"reference": Replicate the reference from the registry, if any, and add the title of the document.

Unassigned or reserved values are not present in these modules.

When the iana-routing-types YANG module is updated, a new "revision" statement must be added in front of the existing revision statements.

IANA has added this new note to the "Address Family Numbers" and "Subsequent Address Family Identifiers (SAFI) Parameters" registries:

When this registry is modified, the YANG module iana-routing-types must be updated as defined in RFC XXXX.

6. Security Considerations

This document defines common routing type definitions (i.e., typedef statements) using the YANG data modeling language. The definitions themselves have no security or privacy impact on the Internet, but the usage of these definitions in concrete YANG modules might have. The security considerations spelled out in the YANG specification [RFC7950] apply for this document as well.

7. Acknowledgements

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Ebben Aries, Lou Berger, Qin Wu, Rob Shakir, Xufeng Liu, and Yingzhen Qu.

Thanks to Martin Bjorkland, Tom Petch, Stewart Bryant, and Radek Krejci for comments on the model and document text. Thanks to Jeff

Haas and Robert Raszuk for suggestions for additional common routing types.

8. References

8.1. 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>>.
- [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>>.
- [IANA-ADDRESS-FAMILY-REGISTRY]
"IANA Address Family Registry",
<<https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml#address-family-numbers-2>>.
- [IANA-SAFI-REGISTRY]
"IANA Subsequent Address Family Identities (SAFI) Parameters Registry", <<https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml#safi-namespace-2>>.

8.2. Informative References

- [IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2008, August 2008.
- [I-D.ietf-bfd-yang]
Rahman, R., Zheng, L., Jethanandani, M., Networks, J., and G. Mirsky, "YANG Data Model for Bidirectional Forwarding Detection (BFD)", draft-ietf-bfd-yang-06 (work in progress), June 2017.

[I-D.ietf-idr-bgp-model]

Shaikh, A., Shakir, R., Patel, K., Hares, S., D'Souza, K., Bansal, D., Clemm, A., Zhdankin, A., Jethanandani, M., and X. Liu, "BGP Model for Service Provider Networks", draft-ietf-idr-bgp-model-02 (work in progress), July 2016.

[I-D.ietf-ospf-yang]

Yeung, D., Qu, Y., Zhang, Z., Chen, I., and A. Lindem, "Yang Data Model for OSPF Protocol", draft-ietf-ospf-yang-08 (work in progress), July 2017.

[I-D.ietf-pim-yang]

Liu, X., McAllister, P., Peter, A., Sivakumar, M., Liu, Y., and f. hu, "A YANG data model for Protocol-Independent Multicast (PIM)", draft-ietf-pim-yang-10 (work in progress), September 2017.

[I-D.ietf-teas-yang-rsvp]

Beeram, V., Saad, T., Gandhi, R., Liu, X., Bryskin, I., and H. Shah, "A YANG Data Model for Resource Reservation Protocol (RSVP)", draft-ietf-teas-yang-rsvp-07 (work in progress), March 2017.

[I-D.ietf-teas-yang-te]

Saad, T., Gandhi, R., Liu, X., Beeram, V., Shah, H., and I. Bryskin, "A YANG Data Model for Traffic Engineering Tunnels and Interfaces", draft-ietf-teas-yang-te-08 (work in progress), July 2017.

[I-D.ietf-bess-l2vpn-yang]

Shah, H., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", draft-ietf-bess-l2vpn-yang-07 (work in progress), October 2017.

[I-D.ietf-bess-l3vpn-yang]

Jain, D., Patel, K., Brissette, P., Li, Z., Zhuang, S., Liu, X., Haas, J., Esale, S., and B. Wen, "Yang Data Model for BGP/MPLS L3 VPNs", draft-ietf-bess-l3vpn-yang-01 (work in progress), April 2017.

[I-D.ietf-mpls-base-yang]

Raza, K., Gandhi, R., Liu, X., Beeram, V., Saad, T., Bryskin, I., Chen, X., Jones, R., and B. Wen, "A YANG Data Model for MPLS Base", draft-ietf-mpls-base-yang-05 (work in progress), July 2017.

- [RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001, <<https://www.rfc-editor.org/info/rfc3032>>.
- [RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, DOI 10.17487/RFC3209, December 2001, <<https://www.rfc-editor.org/info/rfc3209>>.
- [RFC3471] Berger, L., Ed., "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description", RFC 3471, DOI 10.17487/RFC3471, January 2003, <<https://www.rfc-editor.org/info/rfc3471>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February 2006, <<https://www.rfc-editor.org/info/rfc4364>>.
- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <<https://www.rfc-editor.org/info/rfc4664>>.
- [RFC5701] Rekhter, Y., "IPv6 Address Specific BGP Extended Community Attribute", RFC 5701, DOI 10.17487/RFC5701, November 2009, <<https://www.rfc-editor.org/info/rfc5701>>.
- [RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <<https://www.rfc-editor.org/info/rfc5880>>.
- [RFC7274] Kompella, K., Andersson, L., and A. Farrel, "Allocating and Retiring Special-Purpose MPLS Labels", RFC 7274, DOI 10.17487/RFC7274, June 2014, <<https://www.rfc-editor.org/info/rfc7274>>.
- [RFC7432] Sajassi, A., Ed., Aggarwal, R., Bitar, N., Isaac, A., Uttaro, J., Drake, J., and W. Henderickx, "BGP MPLS-Based Ethernet VPN", RFC 7432, DOI 10.17487/RFC7432, February 2015, <<https://www.rfc-editor.org/info/rfc7432>>.

Authors' Addresses

Xufeng Liu
Jabil
8281 Greensboro Drive, Suite 200
McLean VA 22102
USA

EMail: Xufeng_Liu@jabil.com

Yingzhen Qu
Futurewei Technologies, Inc.
2330 Central Expressway
Santa Clara CA 95050
USA

EMail: yingzhen.qu@huawei.com

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

EMail: acee@cisco.com

Christian Hopps
Deutsche Telekom

EMail: chopps@chopps.org

Lou Berger
LabN Consulting, L.L.C.

EMail: lberger@labn.net

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

R. Wilton, Ed.
Cisco Systems
July 3, 2017

Interface Properties for YANG Data Models
draft-wilton-netmod-interface-properties-00

Abstract

This document specifies a better way to restrict interface YANG schema nodes to only those types of interfaces that the nodes are applicable to.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. Problem Definition	3
4. Interface Property Identities	4
5. Potential issues and considerations:	4
6. YANG Modules	4
7. IANA Considerations	7
8. Security Considerations	7
9. Acknowledgments	7
10. References	7
10.1. Normative References	7
10.2. Informative References	7
Appendix A. Examples of possible updates to iana-if-types.yang .	7
Appendix B. Example of interface properties usage in ietf- interfaces-common.yang	10
Appendix C. Example of interface properties usage in ietf- interfaces-ethernet-like.yang	15
Appendix D. Example of interface properties usage in ietf- interfaces.yang	17
Author's Address	21

1. Introduction

This document introduces the concept of generic interface property identities in YANG [RFC7950] data models to solve the problem of how to restrict interface configuration and state schema nodes to the appropriate types of interfaces. E.g., it is desirable to restrict MAC address configuration and state schema nodes to only those types of interfaces that use Ethernet framing, and hence for which MAC address configuration may be applicable. This document defines a set of common interface property YANG identities (e.g., Ethernet-like), and proposes that the iana-interface-type identities in iana-if-type.yang are updated to also derive from the interface property identities (a backwards compatible change). Interface based YANG schema nodes can then be made conditional on the interface property identities rather than being conditional on a hardcoded set of IANA interface types. This approach also allows newly defined interface types to reuse published standard interface YANG schema nodes just by deriving from the appropriate interface property identities, and without requiring new revisions of those published standard YANG modules each time a new IANA interface type is defined.

2. Terminology

This document defines the following terms:

- o interface property identity: A YANG identity that defines a particular generic interface related property that generally relates to multiple interface type identities defined in iana-interface-type.yang. An example of such a property is 'Ethernet-like'.

3. Problem Definition

The YANG language offers the "when" statement that can be used to make nodes conditional on some particular X-Path expression.

In the case, of interface related configuration, the when statement can be used to explicitly list all the iana-interface-type identities that an interface's 'type' leaf may take.

However, the current solution is not ideal for several reasons:

- o Explicitly listing the interface types is somewhat unwieldy, particularly in the case that a schema node may apply to many interfaces. When faced with this scenario, YANG model authors sometimes decide that it is better to keep the model simple and just allow the schema node to be used regardless of interface type.
- o Explicitly listing the interface types has the further problem that the model cannot easily be extended to handle new interface types. If a new interface type is introduced then to reuse the same interface related schema nodes requires that all of the applicable model when statements must be updated to reference the new IANA interface type identity.
- o The current solution does not really work for bespoke vendor specific interface types. Even though the proprietary interface types may be defined in the IANA interface type identity list, it is unlikely that it would be acceptable to update a standards based YANG model to reference a proprietary interface type. Requiring bespoke interfaces to use separate similar configuration and state nodes makes the models less generic and more tedious to use by clients.

4. Interface Property Identities

This draft introduces "Interface Property Identities". These are a set of YANG identities that describe properties that could be associated with multiple different types of interface.

The existing IANA interface type identities are then selectively updated to also derive from one or more of these interface property identities. The YANG module upgrade rules in [RFC7950] allow for the interface type identities to derive from additional identities in a fully backwards compatible way.

YANG modules that define interface specific nodes can then use the YANG 1.1 derived-from() macro in the 'when' statement xpath expression to indicate that the node applies to all interfaces that inherit from the specified interface property.

As new IANA interface types are defined they can derive from the interface property identities and hence acquire the interface specific YANG nodes that have been defined in existing YANG modules without requiring any changes beyond the device updating to use the latest iana-if-types YANG file.

Similarly, vendor specific interface types can also inherit from the interface property identities and also acquire access to the appropriate interface configuration nodes.

As required, new interface property identities can also be defined, again in a backwards compatible way.

5. Potential issues and considerations:

- o Would IANA be willing to manage the new interface property types?
- o iana-interface-types.yang would need to be YANG 1.1. Would this be an issue?
- o Need to define a suitable set of base interface properties.
- o Need to then apply the set of base interface properties to appropriate interface types. Coordination between vendors may be required to get a reasonable base coverage.

6. YANG Modules

```
<CODE BEGINS> file "iana-if-property-type@2017-06-27.yang"

module iana-if-property-type {
```

```
namespace "urn:ietf:params:xml:ns:yang:iana-if-property-type";
prefix ianaifp;

organization "IANA";
contact "";
description
  "This YANG module defines YANG identities for IANA-registered
  interface property types.

  This YANG module is maintained by IANA.

  The latest revision of this YANG module can be obtained from
  the IANA web site.

  Requests for new values should be made to IANA via
  email (iana@iana.org).

  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 initial version of this YANG module is part of XXX;
  see the RFC itself for full legal notices.";
reference
  "IANA 'interface property definitions' registry.";

revision 2017-06-27 {
  description
    "Initial revision";
  reference
    "RFC XXX: IANA Interface Property Type YANG Module";
}

identity iana-if-property-type {
  description
    "Base identity from which specific interface property types are
    derived.";
}

identity physical {
  base iana-if-property-type;
  description
```

```
        "This property represents an interface that has a physical
        hardware representation, or is modelled as such.";
    }

    identity virtual {
        base iana-if-property-type;
        description
            "This property represents an interface that has no external
            physical hardware representation, and is used to represent
            interfaces that are created via configuration.";
    }

    identity sub-interface {
        base iana-if-property-type;
        description
            "This property represents an interface that is a child
            interface that is parented to another interface.";
    }

    identity point-to-point {
        base iana-if-property-type;
        description
            "This property represents an interface that is always
            point-to-point, i.e. the interface can only ever be connected
            to a single other interface, and hence broadcast and multicast
            packet statistics do not make sense.";
    }

    identity multicast {
        base iana-if-property-type;
        description
            "This property represents an interface that could have multiple
            end points, e.g. like an Ethernet interface. Such an
            interface could have separate broadcast and multicast packet
            counters.";
    }

    identity ethernet-like {
        base iana-if-property-type;
        description
            "This property represents an interface that is similar in
            behaviour to an Ethernet interface, and uses Ethernet
            framing.";
    }
}

<CODE ENDS>
```


7. IANA Considerations

This draft proposes that IANA also manage a new registry of "interface properties" alongside the existing "interface type" registry, and to

extend the "interface type" registry to also derive from interface properties identities.

8. Security Considerations

This document discusses an approach how to structure interface related YANG schema. It has no security impact on the Internet.

9. Acknowledgments

This document arose from discussions with Martin Bjorklund, Ladislav Lhotka, and Vladimir Vassilev on the Netmod WG alias.

10. References

10.1. Normative References

- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<http://www.rfc-editor.org/info/rfc7224>>.
- [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>>.

10.2. Informative References

- [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>>.

Appendix A. Examples of possible updates to iana-if-types.yang

The example-iana-if-type.yang module illustrates the type of updates that would be made to iana-if-types.yang to make use of interface properties.

```
<CODE BEGINS> file "example-iana-if-type@2017-06-27.yang"
```

```
module example-iana-if-type {  
  yang-version "1.1";  
  namespace "urn:ietf:params:xml:ns:yang:example-iana-if-type";
```

```
prefix ianaift;

import ietf-interfaces {
    prefix if;
}

import iana-if-property-type {
    prefix ianaifp;
}

// Full description, etc elided for clarity.
organization "IANA";
contact "";
description
    "This example module illustrates how iana-if-type.yang could
    be extended to use interface properties.";
reference
    "IANA 'ifType definitions' registry.
    <http://www.iana.org/assignments/smi-numbers>";

revision 2017-06-27 {
    description
        "Initial example of how IANA if type could be extended";
    reference
        "Taken from draft-rwilton-netmod-interface-properties-00";
}

// Previous revision statements elided for clarity.

identity iana-interface-type {
    base if:interface-type;
    description
        "This identity is used as a base for all interface types
        defined in the 'ifType definitions' registry.";
}

// Most identities elided for clarity, some are retained to
// illustrate how the interface properties are defined.
identity other {
    base iana-interface-type;
    description
        "None of the following";
}

identity ethernetCsmacd {
    base iana-interface-type;
    base ianaifp:physical;
    base ianaifp:multicast;
```

```
    base ianaifp:ethernet-like;
    description
        "For all Ethernet-like interfaces, regardless of speed,
        as per RFC 3635.";
    reference
        "RFC 3635 - Definitions of Managed Objects for the
        Ethernet-like Interface Types";
}

identity ieee8023adLag {
    base iana-interface-type;
    base ianaifp:virtual;
    base ianaifp:multicast;
    base ianaifp:ethernet-like;
    description
        "IEEE 802.3ad Link Aggregate.";
}

identity atm {
    base iana-interface-type;
    base ianaifp:physical;
    base ianaifp:multicast;
    description
        "ATM cells.";
}

identity atmSubInterface {
    base iana-interface-type;
    base ianaifp:virtual;
    base ianaifp:sub-interface;
    base ianaifp:multicast;
    description
        "ATM Sub Interface.";
}

identity l2vlan {
    base iana-interface-type;
    base ianaifp:virtual;
    base ianaifp:sub-interface;
    base ianaifp:multicast;
    description
        "Layer 2 Virtual LAN using 802.1Q.";
}

identity pos {
    base iana-interface-type;
    base ianaifp:point-to-point;
    description
```

```

        "Packet over SONET/SDH Interface.";
    }

    identity irb {
        base iana-interface-type;
        base ianaifp:virtual;
        base ianaifp:multicast;
        base ianaifp:ethernet-like;

        description
            "Integrated Routing and Bridging interface, also known as an
             SVI or BVI interface.

             Note, not currently defined in if-type.yang";
    }
}

<CODE ENDS>

```

Appendix B. Example of interface properties usage in ietf-interfaces-common.yang

The ietf-interfaces-common module defines various interface configuration nodes that are applicable to different types of interfaces and hence would benefit from interface properties.

```

<CODE BEGINS> file "example-ietf-interfaces-common@2017-06-27.yang"

module example-ietf-interfaces-common {
    yang-version 1.1;

    namespace
        "urn:ietf:params:xml:ns:yang:example-ietf-interfaces-common";

    prefix if-cmn;

    import ietf-interfaces {
        prefix if;
    }

    import iana-if-type {
        prefix ianaift;
    }

    import iana-if-property-type {
        prefix ianaifp;
    }
}

```

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

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

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

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

  Editor: Robert Wilton
           <mailto:rwilton@cisco.com>";

description
  "Example of using when statements with interface properties";

revision 2017-06-27 {
  description
    "Examples of using when statements with interface properties";

  reference "Internet draft: draft-ietf-netmod-intf-ext-yang-04";
}

feature bandwidth {
  description "<description elided>";
  reference "Section 3.1 Bandwidth";
}

feature carrier-delay {
  description "<description elided>";
  reference "Section 3.2 Carrier Delay";
}

feature dampening {
  description "<description elided>";
  reference "Section 3.3 Dampening";
}

feature loopback {
  description "<description elided>";
  reference "Section 3.5 Loopback";
}

feature configurable-l2-mtu {
  description "<description elided>";
```

```
    reference "Section 3.6 MTU";
}

feature sub-interfaces {
    description
        "This feature indicates that the device supports the
        instantiation of sub-interfaces. Sub-interfaces are defined
        as logical child interfaces that allow features and forwarding
        decisions to be applied to a subset of the traffic processed
        on the specified parent interface.";
    reference "Section 3.7 Sub-interface";
}

feature forwarding-mode {
    description "<description elided>";
    reference "Section 3.8 Forwarding Mode";
}

/*
 * Define common identities to help allow interface types to be
 * assigned properties.
 */

identity loopback {
    description "Base identity for interface loopback options";
}
identity loopback-internal {
    base loopback;
    description "<description elided>";
}
identity loopback-line {
    base loopback;
    description "<description elided>";
}
identity loopback-connector {
    base loopback;
    description "<description elided>";
}

/*
 * Augments the IETF interfaces model with a leaf to explicitly
 * specify the bandwidth available on an interface.
 */
augment "/if:interfaces/if:interface" {
    description
        "Augments the IETF interface model with optional common
        interface level commands that are not formally covered by any
```

```
        specific standard";

// Various features/nodes elided.

/*
 * Various types of interfaces support a configurable layer 2
 * encapsulation, any that are supported by YANG should be
 * listed here.
 *
 * Different encapsulations can hook into the common encaps-type
 * choice statement.
 */
container encapsulation {
    when
        "derived-from-or-self(..if:type, 'ianaifp:ethernet-like') or
        derived-from-or-self(..if:type, 'ianaifp:sub-interface')" {

        description
            "All interface types that can have a configurable L2
            encapsulation";
    /*
     * TODO - Should we introduce an abstract type to make this
     *         extensible to new interface types, or vendor
     *         specific interface types?
     */
    }

    description
        "Holds the OSI layer 2 encapsulation associated with an
        interface";
    choice encaps-type {
        description "Extensible choice of L2 encapsulations";
    }
}

/*
 * Various types of interfaces support loopback configuration,
 * any that are supported by YANG should be listed here.
 */
leaf loopback {
    when "derived-from(if:type, 'ianaifp:physical')" {
        description
            "Applies to all interfaces that derive from the physical
            interface property.";
    }
}

if-feature "loopback";
```

```

    type identityref {
        base loopback;
    }
    description "Enables traffic loopback.";
}

/*
 * Many types of interfaces support a configurable layer 2 MTU.
 */
leaf l2-mtu {
    if-feature "configurable-l2-mtu";
    type uint16 {
        range "64 .. 65535";
    }
    description
        "The maximum size of layer 2 frames that may be transmitted
        or received on the interface (excluding any FCS overhead).
        In the case of Ethernet interfaces it also excludes the
        4-8 byte overhead of any known (i.e. explicitly matched by
        a child sub-interface) 801.1Q VLAN tags.";
}

/*
 * Add generic support for sub-interfaces.
 *
 * This should be extended to cover all interface types that are
 * child interfaces of other interfaces.
 */
augment "/if:interfaces/if:interface" {
    when "derived-from(if:type, 'ianaifp:sub-interface')" {
        description
            "Applies to all interfaces that derive from the Ethernet-like
            interface property.";
    }

    if-feature "sub-interfaces";

    description
        "Add a parent interface field to interfaces that model
        sub-interfaces";
    leaf parent-interface {

        type if:interface-ref;

        mandatory true;
        description
            "This is the reference to the parent interface of this

```



```

        sub-interface.";
    }
}
}
<CODE ENDS>

```

Appendix C. Example of interface properties usage in ietf-interfaces-ethernet-like.yang

The ietf-interfaces-ethernet-like module defines various interface configuration nodes that are applicable to any interfaces that have "Ethernet-like" semantics, and hence would benefit from interface properties.

```

<CODE BEGINS> file "example-ietf-interfaces-ethernet-
like@2017-06-27.yang"

```

```

module example-ietf-interfaces-ethernet-like {
  yang-version 1.1;

  namespace
    "urn:ietf:params:xml:ns:yang:" +
    "example-ietf-interfaces-ethernet-like";

  prefix ethlike;

  import ietf-interfaces {
    prefix if;
  }

  import ietf-yang-types {
    prefix yang;
  }

  import iana-if-property-type {
    prefix ianaifp;
  }

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

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

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

```

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

Editor:   Robert Wilton
          <mailto:rwilton@cisco.com>";

description
  "Example for when statements using interface properties.";

revision 2017-06-27 {
  description
    "Examples of using when statements with interface properties";

  reference "Internet draft: draft-ietf-netmod-intf-ext-yang-04";
}

/*
 * Configuration parameters for Ethernet-like interfaces.
 */
augment "/if:interfaces/if:interface" {
  when "derived-from(if:type, 'ianaifp:ethernet-like')" {
    description
      "Applies to all interfaces that derive from the Ethernet-like
       interface property.";
  }
  description
    "Augment the interface model with configuration parameters for
     all Ethernet-like interfaces";

  container ethernet-like {
    description "Contains configuration parameters for interfaces
               that use Ethernet framing and expose an Ethernet
               MAC layer.";
    leaf mac-address {
      type yang:mac-address;
      description
        "The configured MAC address of the interface.";
    }
  }
}

/*
 * Operational state for Ethernet-like interfaces.
 */
augment "/if:interfaces-state/if:interface" {
  when "derived-from(if:type, 'ianaifp:ethernet-like')" {
    description
      "Applies to all interfaces that derive from the Ethernet-like
```

```

        interface property.";
    }
    description
        "Augments the interface model with operational state parameters
        for all Ethernet-like interfaces.";
    container ethernet-like {
        description
            "Contains operational state parameters for interfaces that use
            Ethernet framing and expose an Ethernet MAC layer.";
        leaf mac-address {
            type yang:mac-address;
            description
                "The operational MAC address of the interface, if
                applicable";
        }

        leaf bia-mac-address {
            type yang:mac-address;
            description
                "The 'burnt-in' MAC address. I.e the default MAC address
                assigned to the interface if none is explicitly
                configured.";
        }

        container statistics {
            description
                "Packet statistics that apply to all Ethernet-like
                interfaces";
            leaf in-drop-unknown-dest-mac-pkts {
                type yang:counter64;
                units frames;
                description "<long description elided>";
            }
        }
    }
}

```

<CODE ENDS>

Appendix D. Example of interface properties usage in ietf-interfaces.yang

Here is an example of how the ietf-interfaces.yang module could have used interface properties to restrict multicast packet statistics to only those interfaces that support it.

<CODE BEGINS> file "example-ietf-interfaces@2017-06-27.yang"

```
module example-ietf-interfaces {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:example-ietf-interfaces";

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

  import "iana-if-property-type" {
    prefix ianaifp;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
  contact "<elided>";
  description
    "Example of when statements for refining interface statistics";

  revision 2017-06-27 {
    description
      "Example of how some interface statistics could make use of
       interface properties";
    reference
      "RFC 7223: A YANG Data Model for Interface Management";
  }
  /*
  * Typedefs
  */
  // interface-ref typedef elided.
  typedef interface-state-ref {
    type leafref {
      path "/if:interfaces-state/if:interface/if:name";
    }
    description
      "This type is used by data models that need to reference
       the operationally present interfaces.";
  }
  /*
  * Identities
  */
  identity interface-type {
    description
      "Base identity from which specific interface types are
       derived.";
  }
}
```

```
// Features elided.
// Configuration tree elided.

/*
 * Operational state data nodes
 */
container interfaces-state {
  config false;
  description
    "Data nodes for the operational state of interfaces.";
  list interface {
    key "name";
    description
      "The list of interfaces on the device.
      System-controlled interfaces created by the system are
      always present in this list, whether they are configured or
      not.";
    leaf name {
      type string;
      description
        "The name of the interface.
        A server implementation MAY map this leaf to the ifName
        MIB object. Such an implementation needs to use some
        mechanism to handle the differences in size and characters
        allowed between this leaf and ifName. The definition of
        such a mechanism is outside the scope of this document.";
      reference
        "RFC 2863: The Interfaces Group MIB - ifName";
    }
    leaf type {
      type identityref {
        base interface-type;
      }
      mandatory true;
      description
        "The type of the interface.";
      reference
        "RFC 2863: The Interfaces Group MIB - ifType";
    }
  }
}

// Various leaves elided.

container statistics {
  description
    "A collection of interface-related statistics objects.";
  leaf discontinuity-time {
    type yang:date-and-time;
    mandatory true;
  }
}
```

```
        description "<description elided>";
    }
    leaf in-octets {
        type yang:counter64;
        description "<description elided>";
        reference
            "RFC 2863: The Interfaces Group MIB - ifHCInOctets";
    }
    leaf in-unicast-pkts {
        type yang:counter64;
        description
            "The number of packets, delivered by this sub-layer to a
            higher (sub-)layer, that were not addressed to a
            multicast or broadcast address at this sub-layer.
            Discontinuities in the value of this counter can occur
            at re-initialization of the management system, and at
            other times as indicated by the value of
            'discontinuity-time'.";
        reference
            "RFC 2863: The Interfaces Group MIB - ifHCInUcastPkts";
    }
    leaf in-broadcast-pkts {
        when "derived-from(if:type, 'ianaifp:multicast')" {
            description
                "Applies to interfaces that inherit the multicast
                interface property.";
        }

        type yang:counter64;
        description
            "The number of packets, delivered by this sub-layer to a
            higher (sub-)layer, that were addressed to a broadcast
            address at this sub-layer.
            Discontinuities in the value of this counter can occur
            at re-initialization of the management system, and at
            other times as indicated by the value of
            'discontinuity-time'.";
        reference
            "RFC 2863: The Interfaces Group MIB -
            ifHCInBroadcastPkts";
    }
    leaf in-multicast-pkts {
        when "derived-from(if:type, 'ianaifp:multicast')" {
            description
                "Applies to interfaces that inherit the multicast
                interface property.";
        }
    }
```

```
type yang:counter64;
description
  "The number of packets, delivered by this sub-layer to a
  higher (sub-)layer, that were addressed to a multicast
  address at this sub-layer. For a MAC-layer protocol,
  this includes both Group and Functional addresses.
  Discontinuities in the value of this counter can occur
  at re-initialization of the management system, and at
  other times as indicated by the value of
  'discontinuity-time'.";
reference
  "RFC 2863: The Interfaces Group MIB -
  ifHCInMulticastPkts";
}
// Remaining counters elided.
}
}
}
```

<CODE ENDS>

Author's Address

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com