WG Working Group
Internet-Draft
Intended status: Informational
Expires: 27 April 2023

N. Davis
Ciena

24 October 2022

Modelling Boundaries
draft-davis-netmod-modelling-boundaries-00

Abstract

Current modelling techniques appear to have boundaries that make representation of some concepts in modern problems, such as intent and capability, challenging. The concepts all have in common the need to represent uncertainty and vagueness. The challenge results from the rigidity of boundary representation, including the absoluteness of instance value and the process of classification itself, provided by current techniques.

When describing solutions, a softer approach seems necessary where the emphasis is on the focus of a particular thing. Intelligent control (use of AI/ML etc.) could take advantage of partial compatibilities etc. if a softer representation was achieved.

The solution representation appears to require

* Expression of range, preference and focus as a fundamental part of the metamodel

* Recursive tightening of constraints as a native approach of the modeling technique

This lead to need to enhance the metamodel of languages that define properties. It appears that the enhancement could be within, as extensions to, and compatible with current definitions.
YANG is a language used to define properties and it appears that YANG is appropriately formed to accommodate such extensions.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-davis-netmod-modelling-boundaries/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:WG@example.com), which is archived at https://example.com/WG.

Source for this draft and an issue tracker can be found at https://github.com/USER/REPO.

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 27 April 2023.

Copyright Notice

Table of Contents

1. Introduction

The essential challenge being considered in this paper is that of statement of partially constrained definition of a thing (property, collection of properties, entity, collection of entities etc.) and of progression through stages of refinement of constraints leading eventually potentially to precise single value forms of refinements of that thing.

The constrained definition of a thing requires the expression of a boundary that surrounds all allowed/possible values for that thing.

The paper will introduce the term "Occurrence" and explain that through the progression, a single occurrence gives rise to multiple occurrences at the next stage of refinement and that this expansion repeats from one stage to the next.

It appears that many aspects of the industries’ problems/solutions require such a progression of gradual refinement and hence statement of partial constraint and occurrence. However, it seems that current languages do not readily accommodate the necessary structures. It is possible to use existing languages, but the realization seems clumsy and bolted on as opposed to inherent to the language.

Considering the apparent prevalence of need for expression of ranges and uncertainty, it seems strange that there should be no readily available language, so part of the purpose of this paper is to stimulate discussion to help find an appropriate existing language. If, as appears to be the case, there is no well-suited language, then the next obvious step is to consider extension of YANG so that it can accommodate the need.

This paper works through an analysis expressed via threads of observation and exploration that are then woven together to form the fabric of the problem and solution.

In an appendix, a sketch of a YANG form of solution is set out to assist in the understanding of the problem. It is anticipated that the YANG form will seed advancements to the YANG language in this area.

1.1. Observation: Terminology

We all recognize the challenge with any terminology. Terms are for communication convenience (they are not fundamental). Unfortunately, each term comes with baggage and each of us has a different understanding of each term. Sometimes these differences are subtle, but sometimes a term spreads across a very wide space.

Each key term used in this document has specific local meaning which the authors attempt to clarify. However, it is probable that the definitions here are too vague to ensure full shared understanding. Ongoing work will be required.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Analysis

The following section introduces concepts and associated terminology and gradually assembles a picture of the needs.

3.1. Specific challenge areas - some terminology

Each of the following require, for any property, expression of range, preference, uncertainty, interdependency etc. The specific challenges will be discussed in following sections.

The solution to the problem appears to need a language that supports expression narrowing of definition such that that language can be applied recursively to form a progression of increasingly narrowed definitions from the very broad to the very specific.

3.1.1. Specification of *Expectation* and *Intention*

Specification of Expectation/Intention is a statement of desired outcome/experience in terms of constraints which includes statement of preference and acceptable value ranges etc.

This has resulted from some negotiation (or imposition) where, in a simple case, a client and provider have formed an agreement and where the client has an Expectation that the agreement will be fulfilled, and the provider has an Intention to fulfil the agreement.

The expression of outcome/experience is as viewed from the outside of the provider solution, i.e., what is exposed by the provider.

Intention is a sub-case of "Specification of *Capability*" (and expectation is a sub-case of "Specification of *Need*" - note that "Need" is not covered in detail in this document).

Related terms

* intent

* service

3.1.2. Specification of *Capability*

This is a statement of opportunity for behaviour to be exhibited which includes statement of possible ranges and interdependencies etc.

The expression of capability of a provider system, presented as that of an opaque component, results from consideration, by the provider, of various potential assemblies of their internal capabilities (e.g., can be considered as a recursion of systems of components), governed by their business purpose etc.

It is becoming increasingly apparent that there is a need for a machine interpretable expression of capability and hence a language for such expression.

Most recently, specific cases of need have been identified as automation solutions in the following areas mature:

* Advertising capability (both at a commercial boundary and for components in a solution)

* Negotiation towards contract (where capability requirement and offer are refined)

* Planning infrastructure buildout (where capability of solution and of components is required)

* Intent solution formation (intent is the result of negotiation, expressing solution capability)

* Any situation where specialized components need to be assembled

3.1.3. Expression of *Partial Visibility* (of state etc.)

Any real environment will suffer imprecision, loss and disruption. Any statement made about properties (behaviour, characteristics, etc.) of things in that environment may not be fully available (temporarily due to some impairment or permanently due to cost/ complexity (the measure is an approximation as it is not practical to measure precisely)).

This leads to the need, for any property, to be able to state absence, probability, uncertainty and vagueness (varying over the lifecycle etc. as will be discussed later).

3.2. Observation: Progressive Narrowing of definition

Traditional modeling tends to lead to a class model (potentially with inheritance), which provides precise definitions of properties etc. (current YANG models are of this form), where each class is realized as instances in the solution and where each instance provides a precise value for each property defined as mandatory in the class etc.

However, what actually appears to happen in many areas of the solution is a process of gradual narrowing of definition where that narrowing takes place in a progression of discrete steps.

Consider the following rough example progression (stages may be omitted/repeated):

* A version of a standard may provide a definition of technology capability.

* A vendor solution may have a narrower capability than the standard, perhaps due to a target price point etc. A vendor may have several solutions, each with a different narrowing

* An application of the vendor solution may have a further narrowing of capability, perhaps due to some combinatorial effect in the deployment

* The use of a vendor capability at a particular point in a structure of a solution may have a further narrowing of capability as a result of need or policy at that point

* At a particular point in a structure of a solution under particular circumstances there may be an even narrower allowed capability, for example under certain environmental conditions the thermal considerations may require the solution to run at low power etc.

* Proof of concept (PoC) of the solution may have an even narrower allowed capability

* An example diagram related to a single use case for a PoC may require an extremely narrow definition

* Where there is delegated control, even the fully refined "instance", perhaps in the single use case for the PoC mentioned above, may be specified in terms of constrained definition as opposed to absolute value.

* Etc.

Traditional Classification and statement of instance specification do not deal with the above. Some constraint mechanisms deal with the above in part, but these are often an afterthought, are clumsy and have significant shortfalls as will be illustrated.

3.3. Observation: Definition expansion

In addition to the recursive reduction discussed above at each level definition may be introduced that did not appear in the previous stage as a result of capability from the intersection with a separate narrowing. For example, the vendor solution may extend with proprietary features not defined in the standard

Further, there will be evolution and growth so the next development of the standard etc. may extend/adjust the statements.

Of course, any definition introduced at any point in the progression can be narrowed at the next stage of the progression.

The ultimate progression is an intertwining of expansion and reduction stages.

3.4. Observation: Expression of capabilities

For any control solution component at any stage of the above intertwined progression, it is necessary to understand the capability and indeed some of the progression.

This requires runtime interpretation of expression, in normalized uniform machine interpretable form, of the capabilities (properties and constraints) exposed by the relevant assembly.

Traditional classification is too blunt a tool for this purpose as will be illustrated.

3.5. Observation: Application of expression of capability

In a control system context, capability expression applies to:

* the controlled system with respect to the exposed model and allowed activities

* the control system and its exposed capabilities (both the control provider and control client)

Each of the above:

* is always an abstraction/view of the underling real system

* applies for any interaction at any boundary

The above may have further depth such, for example:

* the controlled system exposure can be controlled and adjusted

* the control system exposure can be controlled and adjusted

* etc.

The capability descriptions need to detail all deterministic per case variations (not just a broad- brush statement on the model versions supported).

3.6. Observation: Compatibility

The following applies to any interacting entities with respect to any aspect of interaction (e.g., a control system component interacting with another control system component about things that are controlled).

Note that here a component is a conceptual functional construct that has ports through which it can communicate with other components and that encapsulates some functional capability. Generalized component is described in [ONF TR-512.A.2]

Two components are suitably compatible and can interact with respect to a particular application so long as their exposed capabilities have an appropriate/sufficient intersection.

Interaction between Semi-Compatible Entities is possible where:

* Semantic intersection enables a subset of capabilities (resulting in a meaningful capability set).

* Partially mappable expression provides sufficient meaning (some mappings may be approximate and partially ambiguous, but only in areas where the this is not overly relevant)

The result of the intersection is usually a narrower statement of capability than the statement for the two components (it most it can be the full statement). In some cases, the intersection may be the empty set and hence there can be no interaction opportunity.

* Where a feature is preferred but not mandatory, the empty set intersection is acceptable

* Very few properties are fundamentally mandatory, importance is dependent upon specific application and specific interaction within that application.

For any interaction between two components A and B compatibility is determined by the intersection of:

* application interaction semantic

* interface transfer capability

* component A capabilities/needs

* component B capabilities/needs

If any of the mandatory application interaction semantic elements are eliminated, then the interaction cannot be supported. If preferred or optional semantic elements are eliminated, then the interaction can be supported at some degree of degraded capability.

Note that this discussion fragment focusses on the direct interaction and not on the implications for other interactions etc.

Compatibility must be considered over the intertwined lifecycles of the interacting components as each independently evolves in terms of both functional capability and interface expression. This also includes migration of boundaries.

3.7. Observation: Defining the boundary

The general problem identified is the representation of a semantic space by defining its boundary. Clearly, the boundary is itself defined in terms of ranges, however, the boundary is not necessarily defined with absolute range values, and it is not necessarily fixed in time.

The boundary may, for example,:

* change, in position and in precision, through the lifecycle of the thing (as it matures... where it tends to become tighter)

* be interdependent with other boundaries

* have uncertainty in position of boundary and/or limited interest in positioning of boundary (don’t know, somewhere round about here, don’t care, that’s precise enough)

* also have specification (and measurement) of acceptable, degraded and unacceptable positioning (there is also a need to indicate other aspects, e.g., for how long a particular degradation is acceptable or what the degradation costs etc.)

* changes of positioning and precision over time or over stages of lifecycle

* have associated probability (likelihood of a particular positioning) and preference (for a particular position)

The above considerations apply similarly to intent specification, capability specification and partial visibility.

The same challenges also appear in planning and in negotiation where there is a need to state vaguely understood and interdependent properties etc.

Considering lifecycle stages, a property defining a boundary of a thing, e.g., the property acceptable temperature range, may have one defined range at one part of the lifecycle and another defined range at another part of the lifecycle where this variation is known at the time of specification such that the specification needs to include this lifecycle aspect.

The variation may be temporal or may be dependent upon some other variable property.

3.8. Exploration: The nature of the solution

Considering the observations and other considerations above, the solution requires support for a property to

* Be stated in terms of ranges with focusses and complex (potentially fuzzy) boundaries where that statement defines a semantic volume and where the boundary may not be sharp

* Have statements that interrelate it with another property (or properties)

* Have multiple boundary preference levels and/or probability levels where the preference/importance level is per interaction and not an aspect of fundamental property definition

* Be defined in terms of a narrowing of a previously expressed volume (i.e., a further narrowing) where a single point value is a very narrow range (many single values are actually abstractions of complex ranges, e.g. 2Mbit/s is +/-15ppm)

* Be defined in such a way that simple property definitions are not burdened by the structures that enable sophisticated definition (i.e., the expression should be such that the complexity of expression "folds away" for simple statements)

* Use a representation where there is no distinction in expression opportunity between a statement of capability definition, intent definition, actual value etc. such that all expressions are of the same essential form

This is a fundamental change in the nature of the solution... a change in paradigm and metamodel where the properties are defined by complex/fuzzy bounded/focused spaces related to other complex/fuzzy bounded/focused spaces with preferred/probable positions etc.

3.9. Clarification: Complex/Blurry/Fuzzy boundaries in the solution

To illustrate the complex/fuzzy boundaries, partial compatibility and several other aspects, an example of color usage is helpful. Whilst color may not be the most important aspect of the solution in key industries related to this work, it is an easy example to understand. It is suggested here that the same challenge applies to ALL properties at least to some relevant degree.

Consider a request for a physical item that has a color where the requestor, whilst interested in the color is not overly concerned.

Their request for the item may include an expectation on color where that expectation is that the choice of color is not a showstopper but there is a preference for red and if not red then green. The request does not need to include the color and if not included a choice will be made by the provider on some basis outside the interaction.

The provider may not know what red is but may know green and have the item in green which will be appreciated by the client.

Even if the provider does not have green any color will do. In fact, the provider, in its response, need not even say what the color actually is!

However, if the client indicates that the item provided must not be pink, then unless the provider knows what pink is, it cannot satisfy the request and the boundary now has a hard edge, so an aspect of the request is now mandatory.

In this case, assuming the provider does know what pink is, the provider could respond with "the color is not pink" but provide no more details.

In the example above the definition of color was complex/fuzzy to some degree, the providers understanding of pink may not match the client’s exactly and the definition if the boundary between pink and red may be unclear and vary from occasion to occasion.

The color was specified by the client using colors by name as enumerated literals. Now consider that the provider understand color in RGB. It is possible that the color Red is not just 255,0,0, but is actually a range of colors in a volume bounded on the red scale by 255 at one extreme and 240 at the other. Further, red is a complex volume including on its boundary 255,10,10 and 250,8,8 etc.

Now when the client asks for red, the provider can select color in the range defined.

But it may also be the case that the color is not perfect so that there is always a little green and blue somewhere between 2 and 3 on both scales and the red coloration process is inaccurate so that the color produced is somewhere between 253 and 250.

Further the color fades a little over time and in some lights looks a little more bluish. These factors may need to be taken into account (as interactions between properties) if the request is for red and the duration of use is to be 10 years where the usage will be in various different lights.

So, the request for red must be qualified by the above. In a negotiation the requestor may even have broadened their view of red to include some maroon shades in their preference for Red, so that may now be a list of similar colors etc.

The example above illustrates the need for the opportunity to specify range and interrelationship as a fundamental aspect of specification of color. The color attribute needs the opportunity to deal with the above within its scope, not as a pile of arbitrary other properties.

On the measurement side, it may not be possible to distinguish between anything within a range of 255 and 252 red etc. and further if the light level is low the color measurement may dither etc.

In general, when a specific value is specified, e.g., "A" must equal 5, this equates to a fuzzy setting that has hard boundaries.

It is argued here that the above consideration applies to all properties.

3.10. Observation: Artificial Intelligence and uncertainty

As the spread of system automation progresses, the problem becomes increasingly complex. This leads to the necessary expansion of use of AI/ML techniques in the solution.

These techniques deal well with uncertainty, approximation and fuzziness unlike traditional systems that tend to only work as precise coded solutions and absolute values with the occasional specific hack to deal with ranges and approximation.

AI/ML solutions would benefit from the opportunity to express range, uncertainty etc. in any/all values/structures and to see uncertainty in all inputs. Considering that the problem space is one of range, preference, approximation as discussed, it seems fundamentally necessary to expand the opportunity for expression as discussed in this document.

3.11. Observation: No longer instance config, everything is expectation-intention

The idea of Expectation/Intention as discussed earlier can be further developed. Consider an example where a client has an expectation that the integer property "A" (perhaps a temperature of an "oven" containing a component that needs to operate within a particular range) is to take a value between 5 and 10 (with some unit, e.g., Celsius. It is OK to leave the units open when there are no specific values, but once values are being expressed, the units MUST be provided) . The provider could have the intention to make A take the value between 5 and 10 as requested, but to achieve this a complex process needs to be performed so it will take time to achieve a value in the range and there is some progression that needs to be reported.

Eventually the provider achieves a value in the target range, but it is unable to state the value precisely as there is high-rate jitter and hence it can report that the value is between 6 and 9.

The above example reflects the need to be able to state and report ranges for a property.

Now consider the case where the system is more precise. The client requires and has the expectation for A to take the value 5 (which is simply a very narrow range from 5 to 5) and the provider has the intention to achieve this, but again this will take time. The provider reports progress towards 5 and eventually reports that 5 has been achieved.

The above example reflects the need to be able report convergence on a property value even where the value is simple. In general, the client may want to state a maximum time allowed to achieve any specific outcome and/or the provider may want to state a predicted time for any specific interaction.

Finally consider a case where the system has greater performance. The client requires and has the expectation for A to take the value 5 and the provider the intention to achieve this. The value can be achieved immediately, so the provider can simply report this back directly. In this case the provider would predict an effective delay of 0 seconds (which can be implied as the value is returned immediately).

The final case could be viewed as a SET the property of an instance of a class and hence special, but it is no less of an intent-expectation case than any of the others. Indeed, it is possible that for a particular specific intent-expectation, on some occasions the achievement is immediate and on others it takes a while and for some parts of the range of possible settings the value is precise, but for other parts it is a range (perhaps at the extreme ends of operation).

Clearly, it is highly beneficial (and even arguably, necessary) to have one uniform representation that caters for all cases. Ideally, this method would appear as light as a SET where the value is precise and the achievement is immediate but would deal with the sophistication required where the value is a range, the result is a sub-range, and it takes time to achieve the result.

Assuming that such a representation is achieved, then a traditional "instance specification" is actually sub-case of intention/ achievement (or "intent" as defined by rough common usage) and hence not something distinct. Indeed, the notion is that an instance is simply an occurrence at the most extreme narrowing, the lowest and most detailed available view, of definition and as noted above, this lowest available (visible) view of a realization may not be precise. There are always many details "below" this "lowest available visible view" that are not exposed.

Any expectation/intention statement expression may have a mix of degrees of tightness of statement from vague to single value (and hence suitable to use for all cases of "instance specification") and allow representation of a mix of ranges and of single values.

3.12. Observation: Intention-Expectation interaction

Clearly, a solution does not operate on a single requirement in isolation, there may be multiple agreements and hence multiple Intention-Expectations competing for the solution resources. Within the expression of each Intention-Expectation there is a need to state importance and this will interact with preemption policy.

3.13. Observation: Instance state

As discussed above, an instance is an occurrence at the lowest and most detailed available view at the extreme end of the narrowing. In addition to state related to progression of achievement of expectation/intention (traditional "config"), there is also state related to monitored/measured properties of the solution (not directly related to config).

These properties are derived from monitoring devices that perform some processing of events within the solution. The events are detected by a detector. Very few of all of the possible event sources are monitored by detectors.

All detectors are ultimately imprecise and may fail to operate. The information from a detector may be temporarily unavailable, delayed, degraded etc.

The representation of the simple detected value should include qualifications related to its quality etc.

A machine interpretable specification of capability for the property should provide details of its derivation from other less abstracted properties. For example, there may be a property that is detected where the detections are counted over some period and compared with a threshold
where the crossing of that threshold is reflected by another property that is itself counted and compared with another threshold that if crossed changes the state of the property of concern. An example of a property resulting from this pattern is the Severely Error Second alert.

Understanding this pattern and other related patterns would enable a control solution to interpret the relationship between the various properties (currently, at best, solutions are explicitly coded to deal with properties with human oriented similarities).

3.14. Observation: Foldaway complexity

It was noted in an earlier section that "Ideally, this method would appear as light ... where the value is precise and the achievement is immediate but would deal with the sophistication required where the value is a range, the result is a sub-range, and it takes time to achieve the result.".

In general, it is highly desirable for the representation of common and simple cases to look/be simple and not be burdened by the more sophisticated structures that allow for more complex cases. Ideally the representation has "foldaway complexity".

An analogy can be drawn with human language grammar where the structure that allows sophisticated speech is not visible in simple speech.

Several sketches (in rough JSON) of a configuration statement for a property "temperature" follow.

Basic:

```
  "temperature":"5",
```

More versatile:

```
  "temperature":{
        "acceptable-range":"5-12",
        "preferred-range":"7-9"
  }
```

More sophisticated:

```
  "temperature":{
        "acceptable-range":"5-12",
        "preferred-range":"7-9",
        "upper-warn-threshold":"11",
        "lower-warn-threshold":"6",
        "Fail-alarm"{
              "less-than":"5",
              "greater-than":"12"
        }
  }
```

In this example the schema for:

```
"temperature":{

      "acceptable-range":"5-12",

      "preferred-range":"7-9"

}
```

would identify preferred-range as optional, would identify "acceptable-range" as mandatory and the primary property and would identify the foldaway nature if only one value is provided in the range:

```
"temperature":"6"
```

is conformant with the schema.

In addition, in a simple case a subset schema could be designed that was compatible with the main schema that only allowed the single value temperature.

Ideally, considering the common requirements across all properties, the terms used in the schema nested within the property name would be standard terms etc.

3.15. Exploration: Focusses, boundaries and partial descriptions

Considering the progressive narrowing of boundaries, it is possible to consider anything as represented by a fully capable generalized thing with constraints (everything is a focused thing). It is also possible to consider a subset of things where the focus is thing with ports. Not all things have ports. A thing with one or more ports can be called a component. The component is a thing which has ports. Anything that has ports is a component. The boundary around this focus is the presence of ports.

A physical thing is a thing that can be measured with a ruler. Some, but not all, things that are physical that can be measured with a ruler have ports.

A functional thing is a thing that has behavior. A functional thing is not physical, although it must be realized eventually by physical things. Functional things have ports, as this is where the behavior is exposed (although they need not be represented).

So, there is a set of physical things disjoint from a set of functional things and a set of components that has an intersection with both the physical thing set and the functional thing set where the functional thing set is a subset of the component set.

Anything that has ports is a component (as per the component-system patter described in [ONF TR-512.A.2]). Many components will not yet be known etc., but the semantic of "component" remains unchanged when they are exposed. As not all components are known, not all properties that can apply to components are known. Adding properties does not change the semantics of "component", but it does improve the clarity.

The progression above is essentially, specialization by narrowing of focus. The broadest "Thing" has all possible characteristics and capabilities. Specific semantics relate to the model of a specific thing where that is the narrowing of the broadest thing. The definitions do NOT need to be orthogonal/disjoint.

Consider the thing "Termination". The Termination:

* Covers all aspects of "carrier" signal processing

* includes recursive definition of encapsulated forwarding

* includes all possible properties of termination for any signal (including those not yet defined)

* includes capabilities to extract signal content and further adapt to anther signal

* etc.

3.16. Observation: Two distinct perspectives and viewpoints

Considering a system taking a provider role, there are two distinct perspectives

The external perspective (the effect) - "exposed"

* Capability (advertised to enable negotiation and selection)

* Intention (the agreement resulting from the selection at the end of negotiation)

* Achievement of intention

The internal perspective (the realization)

* Realizations (alternative system design approaches to achieve exposed capabilities)

* Specific chosen realization (the system to be deployed)

* Actual realization achievement

Both perspectives are expressed using the same model pattern and model elements, i.e., the component-system pattern, where a component is described in terms of a system of components and components are specialized as appropriate (as discussed earlier).

There are two viewpoints:

* that of the client where only the external perspective is available

* that of the provider where both the internal (which is private) and external perspective are available

The provider also has control of the mapping between internal and external perspective.

Note that:

* the provider may have distinct roles where each has access to a subset of the provider viewpoint.

* the perspectives and viewpoints apply to both the capabilities being controlled by the system and the capabilities supporting the control system (which are controlled by another system).

* the external perspective relates to "Customer Facing Service" (TM Forum, what is exposed to the customer) and the internal perspective to "Resource Facing Service" (how it is realized (roughly))

* At any arbitrary demarcation, the same approach may be applied

* The actual chosen demarcation may shift as the solution is developed and evolves

* This can be stated as how it looks from the outside and how it looks on the inside

This is discussed further in [ONF TR-512.8] where it is noted that a client talks through a provider port to the provider functions about the controlled system where that controlled system is presented as a projection that has been mapped to an appropriate abstraction of the underlying detail.

3.17. Observation: Capability in more detail

A Capability statement is the statement of visible effect of a thing and is not a statement of the specific realization of that thing. The visible effect may be complex. The thing may have many ports and activity at one port may affect activity at another. Capability statements will include performance and cost (environmental footprint etc.).

It is important to recognized that the statement of capability is NOT exposing intellectual property related to how the capability is achieved.

Expression of externally visible capability and expression of realization of that capability can both use a model of the same types of things, but the specific arrangement will often be very different as the externally visible capability is a severe abstraction of the internal realization where a subset of the capabilities of the underlying system are offered potentially in different terminology and in a different name space.

The approach of expression of capability can be applied recursively at all levels of control abstraction from deep within the device to the most abstract business level.

3.18. Observation: Occurrence

As system is constructed from components. Often a system will make repeated use of the same type of component. Whilst in a realization of that system the components are considered as instances, in the design, they are clearly not instances. But there are many. The term "Occurrence" has been used in ONF work (see [ONF TR-512]).

In a component-system approach, an Occurrence is a single use of a particular component type in a system design structure. There may be many uses of that type in that system design structure, and hence many Occurrences, where each use of that type may have subtly different narrowing of capabilities to each other use and certainly different interconnectivity.

Capability, intent and realization are all specified in terms of system structures and hence all require the use of Occurrence.

Considering the "progressive narrowing" observation earlier in this document, what is a singular thing at one level may result in several separate things at the next level, each of which is a slightly different narrowing of the definition of the original singular thing. These things are Occurrences.

Hence, the progressive narrowing starts with a single Occurrence of thing at the first level and splits this into multiple Occurrence at the next and then each may split at the next etc.

Note that:

* the pictures of devices in a network structure example diagram are essentially Occurrences.

* the presentation of an instance in a management view can be argued to also be an Occurrence.

* that through each progressive narrowing of definition, what was a single "type" at one level of narrowing may cause many "occurrences" at the next level.

* a type is simply an Occurrence with a particular definition where that Occurrence is used in the next level of definition as a type.

3.19. Observation: One model

From a model perspective there appears to be no relevant distinction between what can be requested and what can be achieved. A single model representation of the things and their effect, based upon the recursive/fractal component-system pattern appears appropriate.

The intention/expectation is expressed in terms of a structure of occurrence and what is realized is in terms of a similar structure of occurrence (where at the extreme the structure is exactly the same).

There are however several stages and consequent perspectives:

* the original request (the expectation retained by the client)

* the accepted request (the intention, retained by the provider, normally the same as the expectation)

* the achievable outcome (expressed by the provider, normally the same as the intention)

* the current realization (more precise than the intention)

* the effects of the realization in the perspective of the original request (achievement)

* the difference between the client expectation and the achievement (discrepancy)

For example:

* the client may wish to request an E-Tree with particular characteristics including endpoints, bandwidth, temporal characteristics etc.

* the provider may accept this minus one endpoint.

* the provider may not be able to achieve the accepted request initially as some hardware is not yet in place

* the realization will provide subordinate details.

* the effect of the realization can be abstracted back, freed from some irrelevant detail, to form the achievement (reflecting the details of the original E-Tree request)

* the provider can represent the differences between the original request and the achievement

For all of the above, the key model elements are a multi-pointed connection and a set of terminations. The detail of the realization is supported by a recursion of multi-pointed connections. There is no reason for different representational forms of the different stages of development.

3.20. Observation: Partially satisfied request

The original request from the client may not be satisfiable

* during the progression of activities formulating the solution and acting on that formulation

* initially, although it may be later

* at some intermediate stage in the lifecycle, although it was initially and will be again shortly

* ever

Where there is knowledge of a temporary shortfall, expression of what is achievable as a lifecycle of related statements appears necessary and beneficial. Parts of this lifecycle appear to be definable within individual properties using the mechanism in the metamodel suggested by the various observation here.

3.21. Observation: Other solution elements that benefit

The progressive narrowing approach that yields levels of Occurrences where those Occurrences are defined in terms of semi-constrained properties (as discussed in this document) appears beneficial in the construction of:

-Policy (as per general definition): The condition statement could benefit from a generalized metamodel approach to range etc.

* Profile/Template (for all the various interpretations of these terms): Various methods that support the specification of constraints in a single statement to be applied multiple instances simultaneously.

The constraint statement would benefit modeling in general. For example, in UML, OCL is an add-on that tends to be "beyond" the normal model. An advancement to the essential metamodel would inherently include interaction constraints etc.

3.22. Observation: Outcome and Experience

The term Outcome is being used here to relate to the apparent/ emergent structure/capability made available by the provider and the ongoing behavior of that structure/capability.

The term Experience is being used here to relate to the client perception of the effect that the provider Outcome has (i.e., the client perception of the Outcome). It can also be argued that the Experience is the client Outcome and that the provider Experience includes the feedback by the client on their overall experience of the provided capability etc.

An Outcome/Experience may be a:

* Momentary event of set of events

* Constant state or set of states over time (first order)

* Constant change of state or set of state changes over time (second order)

* ... (nth order)

* Change of state defined some algorithm or set of changes of state defined by a set of algorithms

* Etc.

Both outcome and experience can be expressed in using the same approach discussed in this document.

In the connectivity example discussed earlier, considering the client:

* the expectation for the Outcome is an E-Tree (a resource!)

* the Experience is effect of the E-Tree which is complex apparent adjacency (the true "service", a change of apparent proximity).

3.23. Observation: Metamodel v Model

The metamodel is a model that constrains one or more other model(s). The term metamodel is essentially about the role of the model in the relationship to other models. The model with the metamodel role when related to another model influences that other model and is specifically designed to do so.

A model taking a metamodel role may express how another related model may express properties to be represented.

Clearly a model that takes a metamodel role is just a model and hence may have a related model that takes a metamodel role for it etc.

Note that meta means "providing information about members of its own category" (Merriam Webster).

4. Solution: Formulation

The following subsections consider the observations from the earlier sections and point to aspects of a potential solution.

The first few subsections consider the solution in general independent of specific realization. The final subsection considers the applicability of YANG. Some considerations in the realization independent sections are already supported by YANG, others are not.

4.1. Solution: Methodology

Each type/structure/property is specified in terms of constraints narrowing prior definition/specification. Note that this is a common approach, but the recursive nature is not well represented, and each level of the recursion tends to seem special.

Examples are as discussed earlier:

* A standard may narrow an integer range
* A usage may narrow the standard integer range
* Etc.

The prior definitions must be explicitly (efficiently) referenced. Note that this is a common approach but the specific usage here is distinct from normal usage. Examples relate to earlier discussion:

* A prior definition may be used for several Occurrences in the same specification
* A definition syntax may be transformed, but the semantics cannot be changed, only narrowed
* A property may have preferred values etc.
* Etc.

4.2. Solution: Considering the property

Extending the approach could lead to a more uniform specification of properties in a control system context.

Any property, e.g., temperature, may have combinations of the following features:

* A detector: Allowing opportunity for approximate, unknown, range etc. and allowing notification of change with definable approach to hysteresis etc.
* An associated control: Which has intent, achievement etc. and, especially where it takes time to take the control action, may have some progress on the action etc.
* Thresholds based alerts: Which has intent (as above) and which has an associated state (allowing opportunity for approximate etc.), notification etc.

* Have inter-property interrelationships for any of the above

* Have Units for any of the above

* etc.

4.3. Solution: Occurrence Specification

Any property and its range of opportunities is stated in a specification. Any invariant values in the specification need not be reported in the state of the "instance" (unless the instance is no longer behaving as defined in its specification).

Ideally the metamodel should be such that, when a model designer chooses to define a property, they pick which of the above features are relevant and need not specify each separately. This would lead to automatic name generation etc. where the name structure can be predefined.

A uniform way of expressing each of the above features could be developed as could tooling to generate the representation (e.g., YANG) for each feature given a property name. The application of each feature to a property is essentially the occurrence of the feature.

4.4. Observation: Uniformity of expression

Using a uniform and consistent expression for "occurrences" at all levels of refinement naturally allows for expression of a mix of constraints and absolute values.

4.5. Solution: Tooling support

Clearly, tooling support will be vital for any initiatives in this area to be successful.

5. Target and next steps

There does not seem to be readily available terminology to label/ define the concepts in the problem space

* Hence it has been difficult to discuss what properties the language needs to possess.

* Action: Improve terminology definitions

It appears that there is not a good language suited to solve this problem fully.

* This may only appear to be the case, i.e., there may be a language out there (as it has proved very difficult to describe the problem)

* Action: Continue to explore and refine

It is possible that YANG could evolve to be more suitable

* YANG does not have all the necessary structures or recursion

* Progress sketch for a JSON form of YANG as an illustration of the unification of class and instance statement representation. Work the proposal to suitable maturity (requirements first)

6. Conclusion

Tackling the challenge of modelling of boundaries leads to a more complete method of specification of gradual refinement of definition and of statement of "occurrences" including classes, instances and various forms between.

This method allows for:

* Expression of range, preference and focus as a fundamental part of the metamodel

* Gradual refinement and recursive tightening of constraints as a native approach of the modeling technique

Gradual refinement is required in many areas of the problem/solution space and this more complete method naturally allows for the necessary representation of uncertainty and vagueness. The rigid boundary representation of the current approaches (e.g., class, instance) is accommodated within the method as a narrow case of application of the method.

This softer and more continuous approach to specification refinement with the opportunity for uncertainty, ranges and biases better describes any real-world situation and hence appears more appropriate for an intelligent control solution (using AI/ML etc.) where that solution could take advantage of partial compatibilities etc.

It appears that the enhancements to the language metamodel could be within, as extensions to, and compatible with current definitions of YANG as it appears that YANG is appropriately formed to accommodate such extensions.

Note that the problem appears in expression:

* Intent
* Capability
* Partial Visibility
* Planning
* Negotiation
* Policy
* Profile/Template
* Occurrence
* Etc.

7. Security Considerations

None

8. IANA Considerations

This document has no IANA actions.

9. Informative References

[ONF TR-512] TR-512 Core Information Model (CoreModel) v1.5 at https://opennetworking.org/wp- content/uploads/2021/11/TR-512_v1.5_OnfCoreIm-info.zip (also published by ITU-T as G.7711 at https://www.itu.int/rec/T-REC-G.7711/recommendation.asp?lang=en&parent=T-REC-G.7711-202202-I)

[ONF TR-512.A.2] TR-512.A.2 Appendix: Model Structure, Patterns and Architecture (in Model Description Document within [ONF TR-512])

[ONF TR-512.8]TR-512.8 Control (in Model Description Document within [ONF TR-512])

10. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Appendix A. Appendix A - Problem/Solution Examples

This appendix lists examples (to be expanded).

* A circuit pack with a fixed mapping that supports a narrow subset of the capability defined in the relevant standards...

* A slot in a shelf that takes specific circuit packs

* A temperature sensor with particular range and precision

* A detector that is temporarily absent

* A detector that is degraded due to some temperature issue

* A use of an ethernet termination in a particular configuration of functionality such as the ethernet termination in a G.8032 ring that deals specifically with the termination of signaling traffic

* A request for an e-line where the bandwidth requirement varies over the day

* A request for an e-line where there is an acceptable range of bandwidths and/or a cost profile for bandwidth

* A request for an e-tree where the connectivity requirement varies over the week

* An initial position for planning some infrastructure where the capacity of termination is known and the building is known, but not the specific device

* slicing where the request is for some range of capability and the solution is an approximation to that capability where the approximation is stated as a bounded space.

Appendix B. Appendix B - Sketch of an enhanced YANG form

In this appendix a sketch of a language, that is a development of YANG, that resolves some of the issues is set out. The language is not completely formed, and it is not the intention that this necessarily be the eventual expression, this is simply used for illustrative purposes.

B.1. Progression

Using this language, a progression of increasingly specific models can be set out (as set out in the main body of this document). The refinements at each stage would be in terms of reduction of semantic scope compared to the previous stage. At an early stage of refinement, the component-system pattern emerges.

The language provides definitions for terminology (in a name space) where each term is defined by the specific narrowing (note that the terms are simply a human convenience).

The progression is not a partition. It does not divide the space up into a taxonomy. The progression does lead to sets of capability where those sets may intersect etc.

A traditional model is replaced by what is emergent at a stage in the recursion. A label (in a label space) chosen for a semantic volume should not be reused for anything else. Once defined the label should never be deleted, although it can be obsoleted (i.e., not expected to be used).

If the label is encountered, its definition should be available such that it can be fully interpreted. The label may apply in a narrow application and hence the narrowing definition should be available.

B.2. Language

As noted, it appears that there is not a good language suited to solve this problem fully. This may only appear to be the case, i.e., there may be a language out there, as it has proved very difficult to describe the problem (there does not seem to be readily available terminology for the concepts in the problem space) and hence it has been difficult to discuss what properties the language needs to possess.

To cut through this, the best approach appeared to be to sketch a language and show how it could be applied to hopefully tease out an existing language that could then solve the problem (or so it can be a basis of a new language).

As noted earlier, considering the general and apparently broad extent of the problem, it seems strange that there is not an appropriate machine interpretable language of expression available. It is possible that existing languages can be used to deal with the problem in a somewhat cumbersome way and that this has not yet been observed. Cumbersomeness can be refined out over time. Preferably there will not need to be Yet Another Language!

B.3. Key concepts

Recursive narrowing appears to give rise to "occurrences", where each of a set of occurrences is in part the same as and in part different from each other. Curiously:

* there also appears to be a parallel with the specific approach to specialization taken in ONF (and in YANG models using augment) where a "class" is actually a narrowed "occurrence" of a more general metaclass (see earlier discussion). The distinction in the general thinking is that there are no specific meta-levels. Narrowing can take place through a recursive continuum until all properties have been constrained to absolute precision (which is actually never possible as there is always some uncertainty, rounding etc.).

* When all properties have been constrained the result is the statement of a unique instance at a moment in time (where each occurrence has a lifecycle which intertwines with the lifecycles of other occurrences). But this instance is itself an opaque representation of the effect of underlying detail.

* this approach seems to point to some usages of the term "instance" being flawed and that actually the supposed instance is an "occurrence".

Definitions may be of some type and may cover a subrange of that type. Even in an occurrence that is traditionally called an instance, the property values may be ranges. The key difference for the properties of an instance is that they are unlikely to be further decomposed.

B.4. Observation

At all levels of the recursion there is a mix of schema definition and absolute value (instance values). So, some of the information in a spec looks like instance data and some like schema. This should not be surprising when observing through the lens of recursive narrowing and of occurrence.

Curiously a YANG model definition has instance like data and schema data in it. For example, there is instance like data at the beginning of the definition with things like module, namespace, prefix etc. YANG does not appear uniform in its representation of instance like data and schema data.

The example language developed here attempts to achieve greater uniformity.

B.5. Progressing to the language

Taking JSON as a language of expression of instances and setting out a YANG definition in a JSONized form appeared to allow a uniform blend of instance and schema.

It has been recognized that YIN is a more well-structured form of YANG that points further towards this, and a JSONized form of YIN looked like the hand crafted JSONized YANG that has been constructed here (exploring the expression of recursive narrowing).

JSON has also been simplified here in that every property value is considered as a string and hence in quotes (even numeric quantities). It is not intended that any eventual language is restricted in this way, the approach just simplifies the representation and resultant discussion.

Note that regardless of the form of language chosen, there will be a need to enhance tooling etc. It is intended that the approach should evolve in small backward compatible increments and hence it may be possible to identify value-justified increments in tooling.

B.6. JSONized YANG

The following two snippets show the instance-like header and the schema data in a JSONized form.

B.6.1. JSONised Header

The opening of a YANG module (called a header in this document) is normally of a form illustrated in the example below:

```
module equipment-spec-xyz {

  yang version "1.1";

  namespace "urn:onf:otim:yang:spec-occurrence:equp-spec-xyz{uuid}";

  prefix equip;

etc.
```

In the JSONized form all of the fields are assumed to be "instance" values (where it is assumed that a higher level of specification has specified these). The JSONized form of the example (extended with some other suggested fields) is:

```
"module" : {
```

```
  "name": "equipment-spec-xyz{uuid}"
  "yang-version" : "x.y",
  "namespace" : "urn:onf:otim:yang:spec-occurrence:equp-spec-xyz{uuid}",
  "prefix" : "equip",
  "import" : [
        {
              "name" : "module",
              "prefix" : "mod"
        }
        {
              ....
  "occurrence-encoding" : "JSON",  //Field to explain encoding
  "rule-encoding" : "OCL", //Field to explain encoding
  "utilized-schema": [ //Ref schema at next higher "occurrence" level
        {
              "namespace" : "urn:company:yang:holder-schema-xyz{uuid}",
              "prefix" : "holder"
        }
        {
              "namespace" : "urn:company:yang:tapi-spec",
              "prefix" : "tapi-spec"
        }
        {
              "namespace" : "urn:onf:otcc:yang:tapi-equipment",
```

```
              "prefix" : "tapi-equipment"
              ...
        }
        {
              "namespace" : "urn:onf:otcc:yang:tapi-occurrence",
        ...
        }
        {
              "namespace" : "urn:company:yang:equp-schema-abc{uuid}",
              "prefix" : "equipment"
        }
        ...
  "augment":  [
        {
              "path" : "..."
        }
        ...
```

B.6.2. JSONized body

The core of a YANG module (called a body in this document) is normally of a form illustrated in the example of a fragment below:

```
   grouping connection {

       list connection-end-point {

           uses connection-end-point-ref;

           key 'topology-uuid node-uuid nep-uuid cep-uuid';

           config false;

           min-elements 2;

           description "none";

       }

       list lower-connection {

           uses connection-ref;

           key 'connection-uuid';

   ....
```

In the JSONized form all of the fields are assumed to be "instance" values (where it is assumed that a higher level of specification has specified these). The JSONized form of the example (extended with some other suggested fields) is:

```
       "grouping":{
           "name" : "connection",
           "list": {
               "name" : "connection-end-point",
               "uses" : "connection-end-point-ref",
               "key" : "topology-uuid node-uuid nep-uuid cep-uuid",
               "config" : "false",
               "min-elements" : "2",
               "description" : "none"
           },
           "list": {
               "name" : "lower-connection",
               "uses" : "connection-ref",
               "key" : "connection-uuid",
               "config" : "false",
               "description" : "none"
           },
```

Notice the uniformity/consistency between the representations of the header and the body.

B.7. Schema for the schema

With this a YANG model can define a YANG model (within reason... and probably similar to other self-defining languages - improvements could probably be made to make this more possible).

Considering an extract from tapi-equipment formulated in JSONized YANG:

```
    "grouping":{
          "name":"common-equipment-properties",
          "description":"Properties common to all aspects of equipment",
          "leaf": {
                "name" : "asset-instance-identifier",
                "type" : "string",
                "description" : "none"
          },
          "leaf": {
                "name" : "is-powered",
                "type" : "string",
                "description" : "none"
          },
                "leaf": {
                "name" : "equipment-type-name",
                "type" : "string",
                "description" : "none"
          },
          "leaf": {
                "name" : "manufacture-date",
                "type" : "string",
                "description" : "none"
          },
          "leaf": {
```

```
                "name" : "serial-number",

                "type" : "string",

                "description" : "none"

        },

        "leaf": {

                "name" : "temperature",

                "type" : "decimal64",

                "description" : "none"

        },
```

It is expected that the above model would have been derived from a broader model and that it would reference that model.

In the following development of the model a reference would be provided back to the model above.

This could be used as a general definition, then constrained for a particular application as follows:

```
 "grouping":{

        "name":"common-equipment-properties",

        "description":"Properties common to all aspects of equipment",

        "leaf": {

                "name" : "asset-instance-identifier",

                "type" : "string",

                "pattern" : "^[0-9a-zA-Z]+"$, // A narrowing constraint.

                "description" : "none"

        },

        "leaf": {

                "name" : "is-powered",
```

```
                "type" : "string",
                "supported-constraint" : "NOT_SUPPORTED",//A narrowing.
                "description" : "none"
          },
          "leaf": {
                "name" : "equipment-type-name",
                "type" : "string",
                "description" : "none"
          },
          "leaf": {
                "name" : "manufacturer-date",
                "type" : "string",
                "description" : "none"
          },
          "leaf": {
                "name" : "serial-number",
                "type" : "string",
                "description" : "none"
          },
          "leaf": {
                "name" : "temperature",
                "type" : {
                      "name":"decimal64",
                      "fraction-digits":"1", // A narrowing constraint.
```

```
                "range" : "0.0..100.0",

                "precision":"+0.2,-0.2"

            },

            "units" : "Celcius", // A narrowing constraint.

            "description" : "The temperature of the boiler."

        },
```

Which could be summarized with a reference to the earlier schema and then as follows, where the absent fields are unchanged from the earlier schema and the fields mentioned simply show the change/ addition:

```
     "grouping":{
           "name":"common-equipment-properties",
           "utilized-schema" : "tapi-equipment", //The reference
           "leaf": {
                 "name" : "asset-instance-identifier",
                 "pattern" : "[0-9a-zA-Z]"
           },
           "leaf": {
                 "name" : "is-powered",
                 "supported-constraint" : "NOT_SUPPORTED"
           },
           "leaf": {
                 "name" : "temperature",
                 "fraction-digits": "1",
                 "range" : "0.0... 100.0",
                 "units" : "Celcius"
           },
```

And eventually instance values can be mixed with schema...

```
     "grouping":{
           "name":"common-equipment-properties",
           "description":"Properties common to all aspects of equipment",
           "utilized-schema" : "tapi-equipment",
           "asset-instance-identifier" : "JohnsAsset"
           "leaf": {
```

```
            "name" : "equipment-type-name",
            "type" : "string",
            "description" : "none"
        },
        "leaf": {
            "name" : "manufacturer-date",
            "type" : "string",
            "description" : "none"
        },
        "leaf": {
            "name" : "serial-number",
            "type" : "string",
            "description" : "none"
        },
        "leaf": {
            "name" : "temperature",
            "type" : "decimal64",
            "range" : "0.0... 100.0",
            "units" : "Celcius",
            "description" : "none"
        },
```

Notice that as with normal JSON the name of the property "asset-instance-identifier" is followed by its value (or json-object). The "utilized-schema" has defined "asset-instance-identifier" and the utilization method hence allows the property to either be defined further or narrowed to a fixed value. There are clearly "key words" such as "leaf" that will have been defined in an "earlier" schema, so something like:

```
"field": {

      "name":"leaf",

      "type": "strings"

...
```

And further there would need to be a definition of "name" and "type" etc. and their usage in a structure.

B.8. An example of spec occurrence and rules

This example detail will be added in a later version.

B.8.1. Rough notes

Considering an occurrence of holder in a spec for an equipment, there is a need to identify all compatible equipment types (i.e., that that holder can accommodate). Each type would be associated with a general spec of capability and that spec would relate to the specific application (in the holder) either directly (as the holder is fully capable or the spec is specific to the holder) or via some variables in the spec (that allow modulation of the spec statements).

There are also combinatorial rules. For example, a slot may not be equipped if blocked by a wide card. This can be represented by....Wide card

It is possible that there are multi-slot compatibility rules.

The functional capability may be simply the capability of the equipment type, but there is often functionality that emerges from a combination of hardware.

In this case an equipment may support some fully formed capabilities and some capability fragments that need to be brought together with other fragments to support a meaningful function.

In some cases, functionality from one piece of hardware might compete with functionality from another or functionality might be completely nullified by the presence of another specific piece of hardware.

The equipment-type-identifier is the reference to the equipment spec. This brings details of size and capability.

It is assumed that the holder specification would provide width, position etc. and that there could be a general understanding of size, or there could be a more abstract representation to enable overlap to be accounted for.

B.9. The current schema

This detail will be added in a later version of this document.

B.10. YANG tree

This detail will be added in a later version of this document.

B.11. Instance example

This example detail will be added in a later version of this document.

B.12. The extended schema

This detail will be added in a later version of this document.

B.13. Versioning considerations

This detail will be added in a later version of this document.

Appendix C. Appendix C - My ref / your ref

This appendix will cover "my ref/your ref" naming in relationship to intention/expectation. The detail will be added in a later version of this document.

Appendix D. Appendix D - Occurrence

An "occurrence" at one level of specification is a narrow ("specialized") use of an "occurrence" at the previous higher level of specification. There will be many "occurrences" at a lower level derived from an "occurrence" at a higher level. The "occurrences" at the lower level will be distinct from each other.

Considering the problem space, "Thing" has the broadest spread of semantics covering everything. Function could be considered as a narrowed thing covering only functional aspects and not physical aspects. Termination covers only those functions related to terminating a signal (and not those related to conveying it unchanged.

Considering the formulation of a traditional model, a specific object-class (e.g., Termination Point) is a specific name for an "occurrence" at one level of narrowing ("specialization"). The name object-class is a specific "name" for an "occurrence" at the previous higher level. That previous higher level is called the metamodel in this naming scheme. This is more a narrowing of how to express as opposed to what to express.

Considering the traditional model, "Thing" is effectively an object-class. Considering "Component-System", "Thing" has ports/facets, as do all of its derivatives (unless, of course, they have been narrowed away). The "Component-System" formulation is essentially a narrowing of the expression opportunity in the broadest of problem space considerations at a higher level than "Thing". It effectively provides a quantized representation of a continuous space allowing representation of a refinement of conceptual parts.

In the generalized "occurrence" approach, no specific names are given to the levels and the levels are intentionally not normalized. The approach allows any number of levels, and the number of levels in one "branch" does not need to match the number of levels in another. The approach allows for whatever degrees of gradual refinement are necessary.

So, a traditional "instance" is also an "occurrence" where it is an "occurrence" of specific object- class, i.e., of an "occurrence" at the previous higher level. The "instances" is a leaf in the metamodel structure.

In the generalized "occurrence" approach there is no specific end leaf. Even when the model level has only fully resolved specific values, it is possible to merge in a model with non-specific value "occurrences" and continue to refine.

A specific occurrence:

* Is narrowing of a previously defined occurrence (where there may be many separate distinct narrowings defined at that "level", many occurrences

* Is a mixture of absolute values and definitions

* May be a merge of narrowed previously defined occurrences (where the semantic phase is shifting)

* Is the basis for further narrowing

It also appears that an absolute value of a property at one level may be considered as an unstated approximation of a non-fully resolved property at the next level down. For example, a statement of 2 at one level, refines to 2+/- 15ppm over a short period at the next as the visibility "improves". This seems to say that all levels have a natural uncertainty that may not be known and hence need not be stated.

Appendix E. Appendix E - Narrowing, splitting and merging

E.1. Narrowing

On the most abstract level is "thing" - without any stated parameter, hence without any constraints. "thing" is anything without restriction and can take any shape etc. All possible properties are allowed without restriction (they do have to be declared, but there is no boundary as to what is allowed to be declared). The declared properties are of "thing". It is a semantic set including every possible behavior etc. and all parameters possible.

At the next level of narrowing, some behaviors and hence possible parameters are eliminated and some constrained versions of allowed parameters are exposed. At this point, a convenient name for the specific narrowing can be provided and later references. However, this can also be considered still as "thing".

In the most complete realization, the semantic boundary would be fully defined such that properties on the boundary were appropriately constrained and properties beyond the boundary eliminated. For example, a termination-point cannot have a temperature. So, an expression eliminating this possibility would be necessary and so on for all other properties that cannot be supported. In a more practical implementation, most properties that are beyond the boundary can be assumed to be known to be beyond the boundary and only those with complex constraints need to be stated.

When narrowing "thing", what it can be is reduced and hence so are the parameters that can be exposed. For example, if it is not physical, I cannot expose the parameters for weight.

As the "thing" is narrowed the properties that are allowed to be exposed reduces, but there is a tendency to have more properties exposed as more and more properties become constrained. For example,

color may be irrelevant and hence not exposed or constrained for some of the broader "occurrences", but as the narrowing process approaches the useful definitions, the color becomes constrained and useful, and hence exposed.

Through the narrowing process the set of opportunities becomes smaller and "lighter weight" (possibilities), but more is exposed (semantic mass reduces, semantic visibility increases).

Consider an equipment. It is a physical thing. It may be narrowed to the point where it is constrained such that it can only be plugged into a slot. It is still an equipment, albeit a constrained forms of the general equipment properties for an application. The equipment has a property "requires-slot" set to true. During the first formulation, color may be of no relevance and hence is not exposed. Just because it is not important does not mean that the equipment does not have a color, it means that it is not relevant. It can take any color and I don’t care.

Later, the color becomes important and hence it is exposed and later still it becomes necessary to choose to constrain the allowable colors for an application. It is still an equipment, but it has a spec that constrains what is allowed. The spec is for a narrow form of equipment. It can still be considered as an equipment even though it is a narrow case. It can also be considered as a "thing" with exactly the stated property with some further directly stated constraints.

Narrowing could be considered as pruning (removal of unwanted parts). For example, take a property (leaf/structure in general) from the higher occurrence and potentially:

* Reduce its legal range (perhaps to a single value) of the type. Note that changing the type is allowed if the new type covers the same semantics as the old type. So, Integer to real seems OK and Enum to its corresponding semantic space dimensions seems OK (e.g., color Enum to RGB ranges)

* Specify the units where relevant (or change the units)

* Relate its value to other property values such that its value is constrained

* Remove the property completely

* Change its name (label) to one that represents the narrow version of the broader property, for example, component --> termination-point

This is how modelling is often carried out although it is never formally described as a method. Consider the termination point, the model is for any connection at any layer etc, and there are "profiles" of parameters for particular technologies which augment the termination point. The profiles may may add (actually expose) further parameters for the same technology or for new technologies, but termination point can never have weight as a parameter and certainly cannot be sat on!!

YANG augment is the same process in general, so YANG is positioned appropriately to be the language for this approach.

Interestingly, an AI solution may eventually prefer to use semantics closer to thing than to EthernetTerminationPoint as it can deal with the shades of specification of general things.

E.2. Splitting

Splitting semantics is relatively straight forward. Two distinct occurrences each narrowed from the same higher-level occurrence where the two new occurrences have distinct characteristics.

E.3. Merging

As everything is an "occurrence" of thing, everything has a common highest level of thing. When merging, it may be necessary to go some way back towards that common highest level.

Where the two occurrences are disjoint in distinct characteristics and identical in common characteristics, merging is simply a union. The result may adopt the label of the shared higher level or may have a new label depending upon the labeling (naming) strategy.

Where common characteristics are not identical:

* one may be a simple superset of the other in which case the superset is adopted.

* There may be contradictions in the two specifications in which case there needs to be a simple precedence, e.g., Not overrides Must,

Where merging two (or more) properties from higher models into one property

* There must be a new name for this rephasing of the semantic if neither of the source properties were derived from an origin with the same breadth of space as the new property

* One of the names can be taken if it earlier in the narrowing corresponded to the superset of the new merged result

For example, TerminationPoint narrowed to have no OAM capabilities and then OAM picked up and merged in (again) at some later stage so that this is still TerminationPoint (but it is NOT OAM).

For example, a narrow form of TerminationPoint (processes of traffic at a point) and a narrow form Connection (conveys traffic of space with no transformation) merged into a (strange) long termination that does some distributed processing of traffic. This is neither a TerminationPoint nor a Connection. It could revert to Component with full spec, or it could become a ProcessingSpan (or similar).

Appendix F. Appendix F - A traffic example

This example detail will be added in a later version of this document.

Acknowledgments

Contributors

Martin Skorupski
highstreet technologies
Email: martin.skorupski@highstreet-technologies.com

Author's Address

Nigel Davis
Ciena
Email: ndavis@ciena.com

netmod
Internet-Draft
Intended status: Standards Track
Expires: 27 April 2023

O. G. D. Dios
S. Barguil
Telefonica
M. Boucadair
Orange

24 October 2022

Extensions to the Access Control Lists (ACLs) YANG Model
draft-dbb-netmod-acl-03

Abstract

RFC 8519 defines a YANG data model for Access Control Lists (ACLs). This document discusses a set of extensions that fix many of the limitations of the ACL model as initially defined in RFC 8519.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Network Modeling Working Group mailing list (netmod@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/netmod/.

Source for this draft and an issue tracker can be found at https://github.com/oscargdd/draft-dbb-netmod-enhanced-acl.

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 27 April 2023.

Table of Contents

1. Introduction

[RFC8519] defines Access control lists (ACLs) as a user-ordered set of filtering rules. The model targets the configuration of the filtering behaviour of a device. However, the model structure, as defined in [RFC8519], suffers from a set of limitations. This document describes these limitations and proposes an enhanced ACL structure. The YANG module in this document is solely based on augmentations to the ACL YANG module defined in [RFC8519].

The motivation of such enhanced ACL structure is discussed in detail in Section 3.

When managing ACLs, it is common for network operators to group match elements in pre-defined sets. The consolidation into group matches allows for reducing the number of rules, especially in large scale networks. If it is needed, for example, to find a match against 100 IP addresses (or prefixes), a single rule will suffice rather than creating individual Access Control Entries (ACEs) for each IP address (or prefix). In doing so, implementations would optimize the performance of matching lists vs multiple rules matching.

The enhanced ACL structure is also meant to facilitate the management of network operators. Instead of entering the IP address or port number literals, using user-named lists decouples the creation of the rule from the management of the sets. Hence, it is possible to remove/add entries to the list without redefining the (parent) ACL rule.

In addition, the notion of Access Control List (ACL) and defined sets is generalized so that it is not device-specific as per [RFC8519]. ACLs and defined sets may be defined at network / administrative domain level and associated to devices. This approach facilitates the reusability across multiple network elements. For example, managing the IP prefix sets from a network level makes it easier to maintain by the security groups.

Network operators maintain sets of IP prefixes that are related to each other, e.g., deny-lists or accept-lists that are associated with those provided by a VPN customer. These lists are maintained and manipulated by security expert teams.

Note that ACLs are used locally in devices but are triggered by other tools such as DDoS mitigation [RFC9132] or BGP Flow Spec [RFC8955] [RFC8956]. Therefore, supporting means to easily map to the filtering rules conveyed in messages triggered by these tools is valuable from a network operation standpoint.

2. Terminology

The keywords *MUST*, *MUST NOT*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, *SHOULD NOT*, *RECOMMENDED*, *MAY*, and *OPTIONAL*, when they appear in this document, are to be interpreted as described in [RFC2119].

The terminology for describing YANG modules is defined in [RFC7950]. The meaning of the symbols in the tree diagrams is defined in [RFC8340].

In addition to the terms defined in [RFC8519], this document makes use of the following terms:

* Defined set: Refers to reusable description of one or multiple information elements (e.g., IP address, IP prefix, port number, or ICMP type).

3. Problem Statement & Gap Analysis

3.1. Suboptimal Configuration: Lack of Support for Lists of Prefixes

IP prefix related data nodes, e.g., "destination-ipv4-network" or "destination-ipv6-network", do not support handling a list of IP prefixes, which may then lead to having to support large numbers of ACL entries in a configuration file.

The same issue is encountered when ACLs have to be in place to mitigate DDoS attacks (e.g., [RFC9132] when a set of sources are involved in such an attack. The situation is even worse when both a list of sources and destination prefixes are involved.

Figure 1 shows an example of the required ACL configuration for filtering traffic from two prefixes.

```
{
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "first-prefix",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "my-test-ace",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network":
```

```
                    "2001:db8:6401:1::/64",
                  "source-ipv6-network":
                    "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                },
                "udp": {
                  "source-port": {
                    "operator": "lte",
                    "port": 80
                  },
                  "destination-port": {
                    "operator": "neq",
                    "port": 1010
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }
          ]
        }
      },
      {
        "name": "second-prefix",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "my-test-ace",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network":
                    "2001:db8:6401:c::/64",
                  "source-ipv6-network":
                    "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                },
                "udp": {
                  "source-port": {
                    "operator": "lte",
                    "port": 80
                  },
                  "destination-port": {
                    "operator": "neq",
                    "port": 1010
```

```
                    }
                  }
                },
                "actions": {
                  "forwarding": "accept"
                }
              }
            ]
          }
        }
      ]
    }
  }
```

Figure 1: Example Illustrating Sub-optimal Use of the ACL Model with a Prefix List

Such a configuration is suboptimal for both: - Network controllers that need to manipulate large files. All or a subset for this configuration will need to be passed to the underlying network devices

* Devices may receive such a confirguration and thus will need to maintain it locally.

(Figure 2 depicts an example of an optimized structure:

```
{
  "ietf-access-control-list:acls": {
    "acl": [
      {
        "name": "prefix-list-support",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "my-test-ace",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network": [
                    "2001:db8:6401:1::/64",
                    "2001:db8:6401:c::/64"
                  ],
                  "source-ipv6-network":
                    "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                },
                "udp": {
                  "source-port": {
                    "operator": "lte",
                    "port": 80
                  },
                  "destination-port": {
                    "operator": "neq",
                    "port": 1010
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }
          ]
        }
      }
    ]
  }
}
```

Figure 2: Example Illustrating Optimal Use of the ACL Model in a Network Context.

3.2. Manageability: Impossibility to Use Aliases or Defined Sets

The same approach as the one discussed for IP prefixes can be generalized by introduing the concept of "aliases" or "defined sets".

The defined sets are reusable definitions across several ACLs. Each category is modelled in YANG as a list of parameters related to the class it represents. The following sets can be considered:

* Prefix sets: Used to create lists of IPv4 or IPv6 prefixes.

* Protocol sets: Used to create a list of protocols.

* Port number sets: Used to create lists of TCP or UDP port values (or any other transport protocol that makes uses of port numbers). The identity of the protocols is identified by the protocol set, if present. Otherwise, a set applies to any protocol.

* ICMP sets: Uses to create lists of ICMP-based filters. This applies only when the protocol is set to ICMP or ICMPv6.

A candidate structure is shown in Figure 3:

```
+--rw defined-sets
|  +--rw prefix-sets
|  |  +--rw prefix-set* [name]
|  |     +--rw name        string
|  |     +--rw ip-prefix*   inet:ip-prefix
|  +--rw port-sets
|  |  +--rw port-set* [name]
|  |     +--rw name    string
|  |     +--rw port*   inet:port-number
|  +--rw protocol-sets
|  |  +--rw protocol-set* [name]
|  |     +--rw name             string
|  |     +--rw protocol-name*   identityref
|  +--rw icmp-type-sets
|     +--rw icmp-type-set* [name]
|        +--rw name     string
|        +--rw types* [type]
|           +--rw type              uint8
|           +--rw code?             uint8
|           +--rw rest-of-header?   binary
```

Figure 3: Examples of Defined Sets.

Aliases may also be considered to managed resources that are identified by a combination of various parameters as shown in the candidate tree in Figure 4. Note that some aliases can be provided by decomposing them into separate sets.

```
|  +--rw aliases
|  |  +--rw alias* [name]
|  |     +--rw name                  string
|  |     +--rw prefix*       inet:ip-prefix
|  |     +--rw port-range* [lower-port]
|  |     |  +--rw lower-port    inet:port-number
|  |     |  +--rw upper-port?   inet:port-number
|  |     +--rw protocol*     uint8
|  |     +--rw fqdn*         inet:domain-name
|  |     +--rw uri*          inet:uri
|  +--rw acls
|     ...
|           +--rw rest-of-header?   binary
```

Figure 4: Examples of Aliases.

3.3. Bind ACLs to Devices, Not Only Interfaces

In the context of network management, an ACL may be enforced in many network locations. As such, the ACL module should allow for binding an ACL to multiple devices, not only (abstract) interfaces.

The ACL name must, thus, be unique at the scale of the network, but the same name may be used in many devices when enforcing node-specific ACLs.

3.4. Partial or Lack of IPv4/IPv6 Fragment Handling

[RFC8519] does not support fragment handling capability for IPv6 but offers a partial support for IPv4 by means of 'flags'. Nevertheless, the use of 'flags' is problematic since it does not allow a bitmask to be defined. For example, setting other bits not covered by the 'flags' filtering clause in a packet will allow that packet to get through (because it won't match the ACE).

Defining a new IPv4/IPv6 matching field called 'fragment' is thus required to efficiently handle fragment-related filtering rules.

3.5. Suboptimal TCP Flags Handling

[RFC8519] supports including flags in the TCP match fields, however that structure does not support matching operations as those supported in BGP Flow Spec. Defining this field to be defined as a flag bitmask together with a set of operations is meant to efficiently handle TCP flags filtering rules.

3.6. Rate-Limit Action

[RFC8519] specifies that forwarding actions can be 'accept' (i.e., accept matching traffic), 'drop' (i.e., drop matching traffic without sending any ICMP error message), or 'reject' (i.e., drop matching traffic and send an ICMP error message to the source). However, there are situations where the matching traffic can be accepted, but with a rate-limit policy. Such capability is not currently supported by [RFC8519].

3.7. Payload-based Filtering

Some transport protocols use existing protocols (e.g., TCP or UDP) as substrate. The match criteria for such protocols may rely upon the 'protocol' under '13', TCP/UDP match criteria, part of the TCP/UDP payload, or a combination thereof. [RFC8519] does not support matching based on the payload.

Likewise, the current version of the ACL model does not support filtering of encapsulated traffic.

3.8. Reuse the ACLs Content Across Several Devices

Having a global network view of the ACLs is highly valuable for service providers. An ACL could be defined and applied following the hierarchy of the network topology. So, an ACL can be defined at the network level and, then, that same ACL can be used (or referenced to) in several devices (including termination points) within the same network.

This network/device ACLs differentiation introduces several new requirements, e.g.:

* An ACL name can be used at both network and device levels.

* An ACL content updated at the network level should imply a transaction that updates the relevant content in all the nodes using this ACL.

* ACLs defined at the device level have a local meaning for the specific node.

* A device can be associated with a router, a VRF, a logical system, or a virtual node. ACLs can be applied in physical and logical infrastructure.

4. Overall Module Structure

4.1. Enhanced ACL

```
module: ietf-acl-enh
  augment /ietf-acl:acls/ietf-acl:acl:
    +--rw defined-sets
       +--rw ipv4-prefix-sets
       |  +--rw prefix-set* [name]
       |     +--rw name           string
       |     +--rw description?   string
       |     +--rw prefix*        inet:ipv4-prefix
       +--rw ipv6-prefix-sets
       |  +--rw prefix-set* [name]
       |     +--rw name           string
       |     +--rw description?   string
       |     +--rw prefix*        inet:ipv6-prefix
       +--rw port-sets
       |  +--rw port-set* [name]
       |     +--rw name    string
       |     +--rw port* [id]
       |        +--rw id                                string
       |        +--rw (port)?
       |           +--:(port-range-or-operator)
       |              +--rw 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 protocol-sets
       |  +--rw protocol-set* [name]
       |     +--rw name        string
       |     +--rw protocol*   union
       +--rw icmp-type-sets
          +--rw icmp-type-set* [name]
             +--rw name     string
             +--rw types* [type]
                +--rw type              uint8
```

```
               +--rw code?             uint8
               +--rw rest-of-header?   binary
     augment /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace
               /ietf-acl:matches:
       +--rw (payload)?
          +--:(prefix-pattern)
             +--rw prefix-pattern {match-on-payload}?
                +--rw offset?       identityref
                +--rw offset-end?   uint64
                +--rw operator?     operator
                +--rw prefix?       binary
     augment /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace
               /ietf-acl:matches/ietf-acl:l3/ietf-acl:ipv4:
       +--rw ipv4-fragment
       |  +--rw operator?   operator
       |  +--rw type?       fragment-type
       +--rw source-ipv4-prefix-list?        leafref
       +--rw destination-ipv4-prefix-list?   leafref
       +--rw next-header-set?                leafref
     augment /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace
               /ietf-acl:matches/ietf-acl:l3/ietf-acl:ipv6:
       +--rw ipv6-fragment
       |  +--rw operator?   operator
       |  +--rw type?       fragment-type
       +--rw source-ipv6-prefix-list?        leafref
       +--rw destination-ipv6-prefix-list?   leafref
       +--rw protocol-set?                   leafref
     augment /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace
               /ietf-acl:matches/ietf-acl:l4/ietf-acl:tcp:
       +--rw flags-bitmask
       |  +--rw operator?   operator
       |  +--rw bitmask?    uint16
       +--rw source-tcp-port-set?
       |       -> ../../../../defined-sets/port-sets/port-set/name
       +--rw destination-tcp-port-set?
               -> ../../../../defined-sets/port-sets/port-set/name
     augment /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace
               /ietf-acl:matches/ietf-acl:l4/ietf-acl:udp:
       +--rw source-udp-port-set?
       |       -> ../../../../defined-sets/port-sets/port-set/name
       +--rw destination-udp-port-set?
               -> ../../../../defined-sets/port-sets/port-set/name
     augment /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace
               /ietf-acl:matches/ietf-acl:l4/ietf-acl:icmp:
       +--rw icmp-set?   leafref
     augment /ietf-acl:acls/ietf-acl:acl/ietf-acl:aces/ietf-acl:ace
               /ietf-acl:actions:
       +--rw rate-limit?   decimal64
```

Figure 5: Enhanced ACL tree

4.2. Defined sets

The augmented ACL structure includes several containers to manage reusable sets of elements that can be matched in an ACL entry. Each set is uniquely identified by a name, and can be called from the relevant entry. The following sets are defined:

* IPv4 prefix set: It contains a list of IPv4 prefixes. A match will be considered if the IP address (source or destination, depending on the ACL entry) is contained in any of the prefixes.

* IPv6 prefix set: It contains a list of IPv6 prefixes. A match will be considered if the IP address (source or destination, depending on the ACL entry) is contained in any of the prefixes.

* Port sets: It contains a list of port numbers to be used in TCP / UDP entries. The ports can be individual port numbers, a range of ports, and an operation.

* Protocol sets: It contains a list of protocol values. Each protocol can be identified either by a number (e.g., 17) or a name (e.g., UDP).

* ICMP sets: It contains a list of ICMP types, each of them identified by a type value, optionally the code and the rest of the header.

4.3. TCP Flags Handling

The augmented ACL structure includes a new leaf 'flags-bitmask' to better handle flags.

Clients that support both 'flags-bitmask' and 'flags' matching fields MUST NOT set these fields in the same request.

Figure 6 shows an example of a request to install a filter to discard incoming TCP messages having all flags unset.

```
{
   "ietf-access-control-list:acls": {
     "acl": [{
       "name": "tcp-flags-example",
       "aces": {
         "ace": [{
           "name": "null-attack",
           "matches": {
             "tcp": {
               "flags-bitmask": {
                 "operator": "not any",
                 "bitmask": 4095
               }
             }
           },
           "actions": {
             "forwarding": "drop"
           }
         }]
       }
     }]
   }
 }
```

Figure 6: Example to Deny TCP Null Attack Messages

4.4. Fragments Handling

The augmented ACL structure includes a new leaf 'fragment' to better handle fragments.

Clients that support both 'fragment' and 'flags' matching fields MUST NOT set these fields in the same request.

Figure 7 shows the content of a POST request to allow the traffic destined to 198.51.100.0/24 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

* "drop-all-fragments" ACE: discards all fragments.

* "allow-dns-packets" ACE: accepts DNS packets destined to 198.51.100.0/24.

```
{
     "ietf-access-control-list:acls": {
       "acl": [
         {
           "name": "dns-fragments",
           "type": "ipv4-acl-type",
           "aces": {
             "ace": [
               {
                 "name": "drop-all-fragments",
                 "matches": {
                   "ipv4": {
                     "ipv4-fragment": {
                       "operator": "match",
                       "type": "isf"
                     }
                   }
                 },
                 "actions": {
                   "forwarding": "drop"
                 }
               },
               {
                 "name": "allow-dns-packets",
                 "matches": {
                   "ipv4": {
                     "destination-ipv4-network": "198.51.100.0/24"
                   },
                   "udp": {
                     "destination-port": {
                       "operator": "eq",
                       "port": 53
                     }
                   },
                   "actions": {
                     "forwarding": "accept"
                   }
                 }
               }
             ]
           }
         }
       ]
     }
   }
```

Figure 7: Example Illustrating Candidate Filtering of IPv4 Fragmented Packets.

Figure 8 shows an example of the body of a POST request to allow the traffic destined to 2001:db8::/32 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

* "drop-all-fragments" ACE: discards all fragments (including atomic fragments). That is, IPv6 packets that include a Fragment header (44) are dropped.

* "allow-dns-packets" ACE: accepts DNS packets destined to 2001:db8::/32.

```
{
 "ietf-access-control-list:acls": {
   "acl": [
     {
       "name": "dns-fragments",
       "type": "ipv6-acl-type",
       "aces": {
         "ace": [
           {
             "name": "drop-all-fragments",
             "matches": {
               "ipv6": {
                 "ipv6-fragment": {
                   "operator": "match",
                   "type": "isf"
                 }
               }
             },
             "actions": {
               "forwarding": "drop"
             }
           },
           {
             "name": "allow-dns-packets",
             "matches": {
               "ipv6": {
                 "destination-ipv6-network": "2001:db8::/32"
               },
               "udp": {
                 "destination-port": {
                   "operator": "eq",
                   "port": 53
                 }
               }
             },
             "actions": {
               "forwarding": "accept"
             }
           }
         ]
       }
     }
   ]
 }
}
```

Figure 8: Example Illustrating Candidate Filtering of IPv6 Fragmented Packets.

4.5. Rate-Limit Traffic

In order to support rate-limiting (see Section 3.6), a new action called "rate-limit" is defined.

(#example_5) shows an ACL example to rate-limit incoming SYNs during a SYN flood attack.

```
{
   "ietf-access-control-list:acls": {
     "acl": [{
       "name": "tcp-flags-example-with-rate-limit",
       "aces": {
         "ace": [{
           "name": "rate-limit-syn",
           "matches": {
             "tcp": {
               "flags-bitmask": {
                 "operator": "match",
                 "bitmask": 2
               }
             }
           },
           "actions": {
             "forwarding": "accept",
             "rate-limit": "20.00"
           }
         }]
       }
     }]
   }
 }
```

Figure 9: Example Rate-Limit Incoming TCP SYNs

5. YANG Modules

5.1. Enhanced ACL

```
<CODE BEGINS> file "ietf-acl-enh@2022-10-24.yang"
module ietf-acl-enh {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-acl-enh";
  prefix enh-acl;

  import ietf-inet-types {
    prefix inet;
    reference
```

```
        "RFC 6991: Common YANG Data Types";
    }
    import ietf-access-control-list {
      prefix ietf-acl;
      reference
        "RFC 8519: YANG Data Model for Network Access
                   Control Lists (ACLs), Section 4.1";
    }
    import ietf-packet-fields {
      prefix packet-fields;
      reference
        "RFC 8519: YANG Data Model for Network Access
                   Control Lists (ACLs), Section 4.2";
    }

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

       Author:    Mohamed Boucadair
                  <mailto:mohamed.boucadair@orange.com>
       Author:    Samier Barguil
                  <mailto:samier.barguilgiraldo.ext@telefonica.com>
       Author:    Oscar Gonzalez de Dios
                  <mailto:oscar.gonzalezdedios@telefonica.com>";
    description
      "This module contains YANG definitions for enhanced ACLs.

       Copyright (c) 2022 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 Revised 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 2022-10-24 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: Extensions to the Access Control Lists (ACLs)
```

```
                YANG Model";
    }

    feature match-on-payload {
      description
        "Match based on a pattern is supported.";
    }

    identity offset-type {
      description
        "Base identity for payload offset type.";
    }

    identity layer3 {
      base offset-type;
      description
        "IP header.";
    }

    identity layer4 {
      base offset-type;
      description
        "Transport header (e.g., TCP or UDP).";
    }

    identity payload {
      base offset-type;
      description
        "Transport payload. For example, this represents the beginning
         of the TCP data right after any TCP options.";
    }

    typedef operator {
      type bits {
        bit not {
          position 0;
          description
            "If set, logical negation of operation.";
        }
        bit match {
          position 1;
          description
            "Match bit.  This is a bitwise match operation
             defined as '(data & value) == value'.";
        }
        bit any {
          position 2;
          description
```

```
            "Any bit.  This is a match on any of the bits in
             bitmask.  It evaluates to 'true' if any of the bits
             in the value mask are set in the data,
             i.e., '(data & value) != 0'.";
          }
        }
        description
          "Specifies how to apply the defined bitmask.
          'any' and 'match' bits must not be set simultaneously.";
      }

      typedef fragment-type {
        type bits {
          bit df {
            position 0;
            description
              "Don't fragment bit for IPv4.
               Must be set to 0 when it appears in an IPv6 filter.";
          }
          bit isf {
            position 1;
            description
              "Is a fragment.";
          }
          bit ff {
            position 2;
            description
              "First fragment.";
          }
          bit lf {
            position 3;
            description
              "Last fragment.";
          }
        }
        description
          "Different fragment types to match against.";
      }

      grouping tcp-flags {
        description
          "Operations on TCP flags.";
        leaf operator {
          type operator;
          default "match";
          description
            "How to interpret the TCP flags.";
        }
```

```
      leaf bitmask {
        type uint16;
        description
          "The bitmask matches the last 4 bits of byte 12
          and byte 13 of the TCP header.  For clarity, the 4 bits
          of byte 12 corresponding to the TCP data offset field
          are not included in any matching.";
      }
    }

    grouping fragment-fields {
      description
        "Operations on fragment types.";
      leaf operator {
        type operator;
        default "match";
        description
          "How to interpret the fragment type.";
      }
      leaf type {
        type fragment-type;
        description
          "What fragment type to look for.";
      }
    }

    grouping payload {
      description
        "Operations on payload match.";
      leaf offset {
        type identityref {
          base offset-type;
        }
        description
          "Indicates the payload offset.";
      }
      leaf offset-end {
        type uint64;
        description
          "Indicates the number of bytes to cover when
           performing the prefix match.";
      }
      leaf operator {
        type operator;
        default "match";
        description
          "How to interpret the prefix match.";
      }
```

```
      leaf prefix {
        type binary;
        description
          "The pattern to match against.";
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl" {
      description
        "add a new container to store sets (prefix
         sets, port sets, etc";
      container defined-sets {
        description
          "Predefined sets of attributes used in policy match
           statements.";
        container ipv4-prefix-sets {
          description
            "Data definitions for a list of IPv4 or IPv6
             prefixes which are matched as part of a policy.";
          list prefix-set {
            key "name";
            description
              "List of the defined prefix sets.";
            leaf name {
              type string;
              description
                "Name of the prefix set -- this is used as a label to
                 reference the set in match conditions.";
            }
            leaf description {
              type string;
              description
                "Defined Set description.";
            }
            leaf-list prefix {
              type inet:ipv4-prefix;
              description
                "List of IPv4 prefixes to be used in match
                 conditions.";
            }
          }
        }
        container ipv6-prefix-sets {
          description
            "Data definitions for a list of IPv6 prefixes
             which are matched as part of a policy.";
          list prefix-set {
            key "name";
```

```
            description
              "List of the defined prefix sets.";
            leaf name {
              type string;
              description
                "Name of the prefix set -- this is used as a label to
                 reference the set in match conditions.";
            }
            leaf description {
              type string;
              description
                "A textual description of the prefix list.";
            }
            leaf-list prefix {
              type inet:ipv6-prefix;
              description
                "List of IPv6 prefixes to be used in match
                 conditions.";
            }
          }
        }
        container port-sets {
          description
            "Data definitions for a list of ports which can
             be matched in policies.";
          list port-set {
            key "name";
            description
              "List of port set definitions.";
            leaf name {
              type string;
              description
                "Name of the port set -- this is used as a label to
                 reference the set in match conditions.";
            }
            list port {
              key "id";
              description
                "Port numbers along with the operator on which to
                 match.";
              leaf id {
                type string;
                description
                  "Identifier of the list of port numbers.";
              }
              choice port {
                description
                  "Choice of specifying the port number or referring
```

```
                   to a group of port numbers.";
                container port-range-or-operator {
                  description
                    "Indicates a set of ports.";
                  uses packet-fields:port-range-or-operator;
                }
              }
            }
          }
        }
        container protocol-sets {
          description
            "Data definitions for a list of protocols which can
             be matched in policies.";
          list protocol-set {
            key "name";
            description
              "List of protocol set definitions.";
            leaf name {
              type string;
              description
                "Name of the protocols set -- this is used as a label to
                 reference the set in match conditions.";
            }
            leaf-list protocol {
              type union {
                type uint8;
                type string; //Check if we can reuse an IANA-maintained module
              }
              description
                "Value of the protocl set.";
            }
          }
        }
        container icmp-type-sets {
          description
            "Data definitions for a list of ICMP types which can
             be matched in policies.";
          list icmp-type-set {
            key "name";
            description
              "List of ICMP type set definitions.";
            leaf name {
              type string;
              description
                "Name of the ICMP type set -- this is used as a label to
                 reference the set in match conditions.";
            }
```

```
            list types {
              key "type";
              description
                "Includes a list of ICMP types.";
              uses packet-fields:acl-icmp-header-fields;
            }
          }
        }
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces"
          + "/ietf-acl:ace/ietf-acl:matches" {
      description
        "Add a new match types.";
      choice payload {
        description
          "Match a prefix pattern.";
        container prefix-pattern {
          if-feature "match-on-payload";
          description
            "Rule to perform payload-based match.";
          uses payload;
        }
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces"
          + "/ietf-acl:ace/ietf-acl:matches/ietf-acl:l3/ietf-acl:ipv4" {
      description
        "Handle non-initial and initial fragments for IPv4 packets.";
      container ipv4-fragment {
        description
          "Indicates how to handle IPv4 fragments.";
        uses fragment-fields;
      }
      leaf source-ipv4-prefix-list {
        type leafref {
          path "../../../../defined-sets/ipv4-prefix-sets/prefix-set/name";
        }
        description
          "A reference to a prefix list to match the source address.";
      }
      leaf destination-ipv4-prefix-list {
        type leafref {
          path "../../../../defined-sets/ipv4-prefix-sets/prefix-set/name";
        }
        description
```

```
          "A reference to a prefix list to match the destination address.";
      }
      leaf next-header-set {
        type leafref {
          path "../../../../defined-sets/protocol-sets/protocol-set/name";
        }
        description
          "A reference to a protocol set to match the next-header field.";
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces"
          + "/ietf-acl:ace/ietf-acl:matches/ietf-acl:l3/ietf-acl:ipv6" {
      description
        "Handles non-initial and initial fragments for IPv6 packets.";
      container ipv6-fragment {
        description
          "Indicates how to handle IPv6 fragments.";
        uses fragment-fields;
      }
      leaf source-ipv6-prefix-list {
        type leafref {
          path "../../../../defined-sets/ipv6-prefix-sets/prefix-set/name";
        }
        description
          "A reference to a prefix list to match the source address.";
      }
      leaf destination-ipv6-prefix-list {
        type leafref {
          path "../../../../defined-sets/ipv6-prefix-sets/prefix-set/name";
        }
        description
          "A reference to a prefix list to match the destination address.";
      }
      leaf protocol-set {
        type leafref {
          path "../../../../defined-sets/protocol-sets/protocol-set/name";
        }
        description
          "A reference to a protocol set to match the protocol field.";
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces"
          + "/ietf-acl:ace/ietf-acl:matches/ietf-acl:l4/ietf-acl:tcp" {
      description
        "Handles TCP flags and port sets.";
      container flags-bitmask {
```

```
        description
          "Indicates how to handle TCP flags.";
        uses tcp-flags;
      }
      leaf source-tcp-port-set {
        type leafref {
          path "../../../../defined-sets/port-sets/port-set/name";
        }
        description
          "A reference to a port set to match the source port.";
      }
      leaf destination-tcp-port-set {
        type leafref {
          path "../../../../defined-sets/port-sets/port-set/name";
        }
        description
          "A reference to a port set to match the destination port.";
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces"
          + "/ietf-acl:ace/ietf-acl:matches/ietf-acl:l4/ietf-acl:udp" {
      description
        "Handle UDP port sets.";
      leaf source-udp-port-set {
        type leafref {
          path "../../../../defined-sets/port-sets/port-set/name";
        }
        description
          "A reference to a port set to match the source port.";
      }
      leaf destination-udp-port-set {
        type leafref {
          path "../../../../defined-sets/port-sets/port-set/name";
        }
        description
          "A reference to a port set to match the destination port.";
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces"
          + "/ietf-acl:ace/ietf-acl:matches/ietf-acl:l4/ietf-acl:icmp" {
      description
        "Handle ICMP type sets.";
      leaf icmp-set {
        type leafref {
          path "../../../../defined-sets/icmp-type-sets/icmp-type-set/name";
        }
```

```
        description
          "A reference to an ICMP type set to match the ICMP type field.";
      }
    }

    augment "/ietf-acl:acls/ietf-acl:acl/ietf-acl:aces"
          + "/ietf-acl:ace/ietf-acl:actions" {
      description
        "Rate-limit action.";
      leaf rate-limit {
        when "../ietf-acl:forwarding = 'ietf-acl:accept'" {
          description
            "Rate-limit valid only when accept action is used.";
        }
        type decimal64 {
          fraction-digits 2;
        }
        units "bytes per second";
        description
          "Indicates a rate-limit for the matched traffic.";
      }
    }
  }
  <CODE ENDS>
```

6. Security Considerations (TBC)

The YANG modules specified in this document define 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 Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [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.

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

* TBC

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:

* TBC

7. IANA Considerations

7.1. URI Registration

This document requests IANA to register the following URI in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

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

7.2. YANG Module Name Registration

This document requests IANA to register the following YANG module in the "YANG Module Names" subregistry [RFC6020] within the "YANG Parameters" registry.

```
name: ietf-acl-enh
namespace: urn:ietf:params:xml:ns:yang:ietf-ietf-acl-enh
maintained by IANA: N
prefix: enh-acl
reference: RFC XXXX
```

8. References

8.1. Normative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/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/rfc/rfc6242>.

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/rfc/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/rfc/rfc8446>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/rfc/rfc8519>.

[RFC8956] Loibl, C., Ed., Raszuk, R., Ed., and S. Hares, Ed., "Dissemination of Flow Specification Rules for IPv6", RFC 8956, DOI 10.17487/RFC8956, December 2020, <https://www.rfc-editor.org/rfc/rfc8956>.

8.2. Informative References

[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/rfc/rfc8340>.

[RFC8955] Loibl, C., Hares, S., Raszuk, R., McPherson, D., and M. Bacher, "Dissemination of Flow Specification Rules", RFC 8955, DOI 10.17487/RFC8955, December 2020, <https://www.rfc-editor.org/rfc/rfc8955>.

[RFC9132] Boucadair, M., Ed., Shallow, J., and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 9132, DOI 10.17487/RFC9132, September 2021, <https://www.rfc-editor.org/rfc/rfc9132>.

Appendix A. Acknowledgements

Many thanks to Jon Shallow and Miguel Cros for the review and comments to the document, incuding priror to publishing the document.

Thanks for Qin Wu for the comments and suggestions.

This work is partially supported by the European Commission under Horizon 2020 Secured autonomic traffic management for a Tera of SDN flows (Teraflow) project (grant agreement number 101015857).

Authors' Addresses

Oscar Gonzalez de Dios
Telefonica
Email: oscar.gonzalezdedios@telefonica.com

Samier Barguil
Telefonica
Email: samier.barguilgiraldo.ext@telefonica.com

Mohamed Boucadair
Orange
Email: mohamed.boucadair@orange.com

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.

Table of Contents

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:

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

- Internal loopback - All egress traffic on the interface is internally looped back within the interface to be received on the ingress path.
- Line loopback - All ingress traffic received on the interface is internally looped back within the interface to the egress path.
- 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:

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

- Update modules from github and tree diagram.

8.2. Version -09

- Fixed IANA section.

8.3. Version -08

- Initial updates after WG LC comments.

8.4. Version -07

- Minor editorial updates

8.5. Version -06

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

8.6. Version -05

- Incorporate feedback from Andy Bierman

8.7. Version -04

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

8.8. Version -03

- Fixed incorrect module name references, and updated tree output

8.9. Version -02

- 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 registred 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.

- /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:

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

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

- /if:interfaces/if:interface/encapsulation
- /if:interfaces/if:interface/max-frame-size
- /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:

- /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:

- 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. Sivaraj, "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
Updates: 6020, 7950, 8407, 8525 (if approved)
Intended status: Standards Track
Expires: 27 April 2023

R. Wilton, Ed.
Cisco Systems, Inc.
R. Rahman, Ed.

B. Lengyel, Ed.
Ericsson
J. Clarke
Cisco Systems, Inc.
J. Sterne
Nokia

24 October 2022

Updated YANG Module Revision Handling
draft-ietf-netmod-yang-module-versioning-07

Abstract

This document specifies a new YANG module update procedure that can document when non-backwards-compatible changes have occurred during the evolution of a YANG module. It extends the YANG import statement with an earliest revision filter to better represent inter-module dependencies. It provides guidelines for managing the lifecycle of YANG modules and individual schema nodes. It provides a mechanism, via the revision-label YANG extension, to specify a revision identifier for YANG modules and submodules. This document updates RFC 7950, RFC 6020, RFC 8407 and RFC 8525.

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 27 April 2023.

Table of Contents

1. Introduction

The current YANG [RFC7950] module update rules require that updates of YANG modules preserve strict backwards compatibility. This has caused problems as described in [I-D.ietf-netmod-yang-versioning-reqs]. This document recognizes the need to sometimes allow YANG modules to evolve with non-backwards-compatible changes, which can cause breakage to clients and importing YANG modules. Accepting that non-backwards-compatible changes do sometimes occur, it is important to have mechanisms to report when these changes occur, and to manage their effect on clients and the broader YANG ecosystem.

This document defines a flexible versioning solution. Several other documents build on this solution with additional capabilities. [I-D.ietf-netmod-yang-schema-comparison] specifies an algorithm that can be used to compare two revisions of a YANG schema and provide granular information to allow module users to determine if they are impacted by changes between the revisions. The [I-D.ietf-netmod-yang-semver] document extends the module versioning work by introducing a revision label scheme based on semantic versioning. YANG packages [I-D.ietf-netmod-yang-packages] provides a mechanism to group sets of related YANG modules together in order to manage schema and conformance of YANG modules as a cohesive set instead of individually. Finally, [I-D.ietf-netmod-yang-ver-selection] provides a schema selection mechanism that allows a client to choose which schemas to use when interacting with a server from the available schema that are supported and advertised by the server. These other documents are mentioned here as informative references. Support of the other documents is not required in an implementation in order to take advantage of the mechanisms and functionality offered by this module versioning document.

The document comprises five parts:

* Refinements to the YANG 1.1 module revision update procedure, supported by new extension statements to indicate when a revision contains non-backwards-compatible changes, and an optional revision label.

* A YANG extension statement allowing YANG module imports to specify an earliest module revision that may satisfy the import dependency.

* Updates and augmentations to ietf-yang-library to include the revision label in the module and submodule descriptions, to report how "deprecated" and "obsolete" nodes are handled by a server, and to clarify how module imports are resolved when multiple revisions could otherwise be chosen.

* Considerations of how versioning applies to YANG instance data.

* Guidelines for how the YANG module update rules defined in this document should be used, along with examples.

Note to RFC Editor (To be removed by RFC Editor)

Open issues are tracked at https://github.com/netmod-wg/yang-ver-dt/issues.

1.1. Updates to YANG RFCs

This document updates [RFC7950] section 11 and [RFC6020] section 10. Section 3 describes modifications to YANG revision handling and update rules, and Section 4 describes a YANG extension statement to do import by derived revision.

This document updates [RFC7950] section 5.2, [RFC6020] section 5.2 and [RFC8407] section 3.2. Section 3.4.1 describes the use of a revision label in the name of a file containing a YANG module or submodule.

This document updates [RFC7950] section 5.6.5 and [RFC8525]. Section 5.1 defines how a client of a YANG library datastore schema resolves ambiguous imports for modules which are not "import-only".

This document updates [RFC8407] section 4.7. Section 7 provides guidelines on managing the lifecycle of YANG modules that may contain non-backwards-compatible changes and a branched revision history.

This document updates [RFC8525] with augmentations to include revision labels in the YANG library data and two boolean leafs to indicate whether status deprecated and status obsolete schema nodes are implemented by the server.

2. Terminology and Conventions

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.

This document makes use of the following terminology introduced in the YANG 1.1 Data Modeling Language [RFC7950]:

* schema node

In addition, this document uses the following terminology:

* YANG module revision: An instance of a YANG module, uniquely identified with a revision date, with no implied ordering or backwards compatibility between different revisions of the same module.

* Backwards-compatible (BC) change: A backwards-compatible change between two YANG module revisions, as defined in Section 3.1.1

* Non-backwards-compatible (NBC) change: A non-backwards-compatible change between two YANG module revisions, as defined in Section 3.1.2

3. Refinements to YANG revision handling

[RFC7950] and [RFC6020] assume, but do not explicitly state, that the revision history for a YANG module or submodule is strictly linear, i.e., it is prohibited to have two independent revisions of a YANG module or submodule that are both directly derived from the same parent revision.

This document clarifies [RFC7950] and [RFC6020] to explicitly allow non-linear development of YANG module and submodule revisions, so that they MAY have multiple revisions that directly derive from the same parent revision. As per [RFC7950] and [RFC6020], YANG module and submodule revisions continue to be uniquely identified by their revision date, and hence all revisions of a given module or submodule MUST have unique revision dates.

A corollary to the above is that the relationship between two module or submodule revisions cannot be determined by comparing the module or submodule revision date alone, and the revision history, or revision label, must also be taken into consideration.

A module's name and revision date identifies a specific immutable definition of that module within its revision history. Hence, if a module includes submodules then to ensure that the module's content is uniquely defined, the module's "include" statements SHOULD use "revision-date" substatements to specify the exact revision date of each included submodule. When a module does not include its submodules by revision-date, the revision of submodules used cannot be derived from the including module. Mechanisms such as YANG packages [I-D.ietf-netmod-yang-packages], and YANG library [RFC8525], MAY be used to specify the exact submodule revisions used when the submodule revision date is not constrained by the "include" statement.

[RFC7950] section 11 and [RFC6020] section 10 require that all updates to a YANG module are BC to the previous revision of the module. This document introduces a method to indicate that an NBC change has occurred between module revisions: this is done by using a new "non-backwards-compatible-revision" YANG extension statement in the module revision history.

Two revisions of a module or submodule MAY have identical content except for the revision history. This could occur, for example, if a module or submodule has a branched history and identical changes are applied in multiple branches.

3.1. Updating a YANG module with a new revision

This section updates [RFC7950] section 11 and [RFC6020] section 10 to refine the rules for permissible changes when a new YANG module revision is created.

Where pragmatic, updates to YANG modules SHOULD be backwards-compatible, following the definition in Section 3.1.1.

A new module revision MAY contain NBC changes, e.g., the semantics of an existing data-node definition MAY be changed in an NBC manner without requiring a new data-node definition with a new identifier. A YANG extension, defined in Section 3.2, is used to signal the potential for incompatibility to existing module users and readers.

As per [RFC7950] and [RFC6020], all published revisions of a module are given a new unique revision date. This applies even for module revisions containing (in the module or included submodules) only changes to any whitespace, formatting, comments or line endings (e.g., DOS vs UNIX).

3.1.1. Backwards-compatible rules

A change between two module revisions is defined as being "backwards-compatible" if the change conforms to the module update rules specified in [RFC7950] section 11 and [RFC6020] section 10, updated by the following rules:

* A "status" "deprecated" statement MAY be added, or changed from "current" to "deprecated", but adding or changing "status" to "obsolete" is a non-backwards-compatible change.

* YANG schema nodes with a "status" "obsolete" substatement MAY be removed from published modules, and the removal is classified as a backwards-compatible change. In some circumstances it may be helpful to retain the obsolete definitions since their identifiers may still be referenced by other modules and to ensure that their identifiers are not reused with a different meaning.

* A statement that is defined using the YANG "extension" statement MAY be added, removed, or changed, if it does not change the semantics of the module. Extension statement definitions SHOULD specify whether adding, removing, or changing statements defined by that extension are backwards-compatible or non-backwards-compatible.

* Any change made to the "revision-date" or "revision-or-derived" substatements of an "import" statement, including adding new "revision-date" or "revision-or-derived" substatements, changing the argument of any "revision-date" or "revision-or-derived" substatetements, or removing any "revision-date" or "revision-or-dervied" substatements, is classified as backwards-compatible.

* Any changes (including whitespace or formatting changes) that do not change the semantic meaning of the module are backwards-compatible.

3.1.2. Non-backwards-compatible changes

Any changes to YANG modules that are not defined by Section 3.1.1 as being backwards-compatible are classified as "non-backwards-compatible" changes.

3.2. Non-backwards-compatible-revision extension statement

The "rev:non-backwards-compatible-revision" extension statement is used to indicate YANG module revisions that contain NBC changes.

If a revision of a YANG module contains changes, relative to the preceding revision in the revision history, that do not conform to the module update rules defined in Section 3.1.1, then a "rev:non-backwards-compatible-revision" extension statement MUST be added as a substatement to the "revision" statement.

Adding, modifying or removing a "rev:non-backwards-compatible-revision" extension statement is considered to be a BC change.

3.3. Removing revisions from the revision history

Authors may wish to remove revision statements from a module or submodule. Removal of revision information may be desirable for a number of reasons including reducing the size of a large revision history, or removing a revision that should no longer be used or imported. Removing revision statements is allowed, but can cause issues and SHOULD NOT be done without careful analysis of the potential impact to users of the module or submodule. Doing so can lead to import breakages when import by revision-or-derived is used. Moreover, truncating history may cause loss of visibility of when non-backwards-compatible changes were introduced.

An author MAY remove a contiguous sequence of entries from the end (i.e., oldest entries) of the revision history. This is acceptable even if the first remaining (oldest) revision entry in the revision history contains a rev:non-backwards-compatible-revision substatement.

An author MAY remove a contiguous sequence of entries in the revision history as long as the presence or absence of any existing rev:non-backwards-compatible-revision substatements on all remaining entries still accurately reflect the compatibility relationship to their preceding entries remaining in the revision history.

The author MUST NOT remove the first (i.e., newest) revision entry in the revision history.

Example revision history:

```
   revision 2020-11-11 {
     rev:revision-label 4.0.0;
     rev:non-backwards-compatible-revision;
   }

   revision 2020-08-09 {
     rev:revision-label 3.0.0;
     rev:non-backwards-compatible-revision;
   }

   revision 2020-06-07 {
     rev:revision-label 2.1.0;
   }

   revision 2020-02-10 {
     rev:revision-label 2.0.0;
     rev:non-backwards-compatible-revision;
   }

   revision 2019-10-21 {
     rev:revision-label 1.1.3;
   }

   revision 2019-03-04 {
     rev:revision-label 1.1.2;
   }

   revision 2019-01-02 {
     rev:revision-label 1.1.1;
   }
```

In the revision history example above, removing the revision history entry for 2020-02-10 would also remove the rev:non-backwards-compatible-revision annotation and hence the resulting revision history would incorrectly indicate that revision 2020-06-07 is backwards-compatible with revisions 2019-01-02 through 2019-10-21 when it is not, and so this change cannot be made. Conversely, removing one or more revisions out of 2019-03-04, 2019-10-21 and 2020-08-09 from the revision history would still retain a consistent revision history, and is acceptable, subject to an awareness of the concerns raised in the first paragraph of this section.

3.4. Revision label

Each revision entry in a module or submodule MAY have a revision label associated with it, providing an alternative alias to identify a particular revision of a module or submodule. The revision label could be used to provide an additional versioning identifier associated with the revision.

A revision label scheme is a set of rules describing how a particular type of revision-label operates for versioning YANG modules and submodules. For example, YANG Semver [I-D.ietf-netmod-yang-semver] defines a revision label scheme based on Semver 2.0.0 [semver]. Other documents may define other YANG revision label schemes.

Submodules MAY use a revision label scheme. When they use a revision label scheme, submodules MAY use a revision label scheme that is different from the one used in the including module.

The revision label space of submodules is separate from the revision label space of the including module. A change in one submodule MUST result in a new revision label of that submodule and the including module, but the actual values of the revision labels in the module and submodule could be completely different. A change in one submodule does not result in a new revision label in another submodule. A change in a module revision label does not necessarily mean a change to the revision label in all included submodules.

If a revision has an associated revision label, then it may be used instead of the revision date in a "rev:revision-or-derived" extension statement argument.

A specific revision-label identifies a specific revision of the module. If two YANG modules contain the same module name and the same revision-label (and hence also the same revision-date) in their latest revision statement, then the file contents of the two modules, including the revision history, MUST be identical.

3.4.1. File names

This section updates [RFC7950] section 5.2, [RFC6020] section 5.2 and [RFC8407] section 3.2

If a revision has an associated revision label, then it is RECOMMENDED that the name of the file for that revision be of the form:

module-or-submodule-name ['#' revision-label] ('.yang' / '.yin')

E.g., acme-router-module#2.0.3.yang

YANG module (or submodule) files may be identified using either the revision-date (as per [RFC8407] section 3.2) or the revision-label.

3.4.2. Revision label scheme extension statement

The optional "rev:revision-label-scheme" extension statement is used to indicate which revision-label scheme a module or submodule uses. There MUST NOT be more than one revision label scheme in a module or submodule. The mandatory argument to this extension statement:

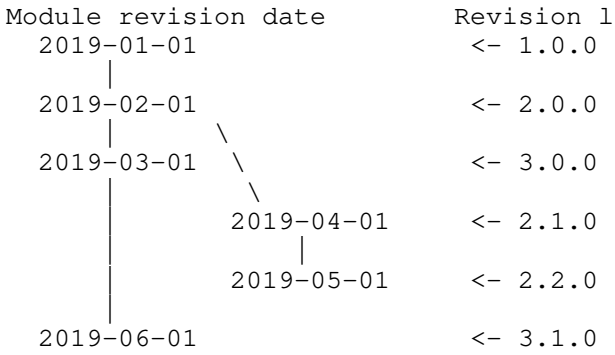
* specifies the revision-label scheme used by the module or submodule
* is defined in the document which specifies the revision-label scheme
* MUST be an identity derived from "revision-label-scheme-base".

The revision-label scheme used by a module or submodule SHOULD NOT change during the lifetime of the module or submodule. If the revision-label scheme used by a module or submodule is changed to a new scheme, then all revision-label statements that do not conform to the new scheme MUST be replaced or removed.

3.5. Examples for updating the YANG module revision history

The following diagram, explanation, and module history illustrates how the branched revision history, "non-backwards-compatible" extension statement, and "revision-label" extension statement could be used:

Example YANG module with branched revision history.



The tree diagram above illustrates how an example module's revision history might evolve, over time. For example, the tree might represent the following changes, listed in chronological order from the oldest revision to the newest revision:

Example module, revision 2019-06-01:

```
module example-module {

  namespace "urn:example:module";
  prefix "prefix-name";
  rev:revision-label-scheme "yangver:yang-semver";

  import ietf-yang-revisions { prefix "rev"; }
  import ietf-yang-semver { prefix "yangver"; }

  description
    "to be completed";

  revision 2019-06-01 {
    rev:revision-label 3.1.0;
    description "Add new functionality.";
  }

  revision 2019-03-01 {
    rev:revision-label 3.0.0;
    rev:non-backwards-compatible-revision;
    description
      "Add new functionality. Remove some deprecated nodes.";
  }

  revision 2019-02-01 {
    rev:revision-label 2.0.0;
    rev:non-backwards-compatible-revision;
    description "Apply bugfix to pattern statement";
  }

  revision 2019-01-01 {
    rev:revision-label 1.0.0;
    description "Initial revision";
  }

  //YANG module definition starts here
}
```

Example module, revision 2019-05-01:

```
module example-module {

  namespace "urn:example:module";
  prefix "prefix-name";
  rev:revision-label-scheme "yangver:yang-semver";

  import ietf-yang-revisions { prefix "rev"; }
  import ietf-yang-semver { prefix "yangver"; }

  description
    "to be completed";

  revision 2019-05-01 {
    rev:revision-label 2.2.0;
    description "Backwards-compatible bugfix to enhancement.";
  }

  revision 2019-04-01 {
    rev:revision-label 2.1.0;
    description "Apply enhancement to older release train.";
  }

  revision 2019-02-01 {
    rev:revision-label 2.0.0;
    rev:non-backwards-compatible-revision;
    description "Apply bugfix to pattern statement";
  }

  revision 2019-01-01 {
    rev:revision-label 1.0.0;
    description "Initial revision";
  }

  //YANG module definition starts here
}
```

4. Import by derived revision

[RFC7950] and [RFC6020] allow YANG module "import" statements to optionally require the imported module to have a particular revision date. In practice, importing a module with an exact revision date is often too restrictive because it requires the importing module to be updated whenever any change to the imported module occurs. The alternative choice of using an import statement without any revision date statement is also not ideal because the importing module may not work with all possible revisions of the imported module.

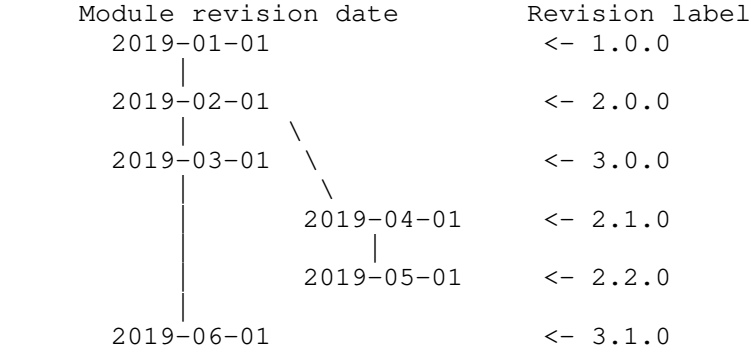
Instead, it is desirable for an importing module to specify a "minimum required revision" of a module that it is compatible with, based on the assumption that later revisions derived from that "minimum required revision" are also likely to be compatible. Many possible changes to a YANG module do not break importing modules, even if the changes themselves are not strictly backwards-compatible. E.g., fixing an incorrect pattern statement or description for a leaf would not break an import, changing the name of a leaf could break an import but frequently would not, but removing a container would break imports if that container is augmented by another module.

The ietf-revisions module defines the "revision-or-derived" extension statement, a substatement to the YANG "import" statement, to allow for a "minimum required revision" to be specified during import:

> The argument to the "revision-or-derived" extension statement is a revision date or a revision label.
>
> A particular revision of an imported module satisfies an import's "revision-or-derived" extension statement if the imported module's revision history contains a revision statement with a matching revision date or revision label.
>
> An "import" statement MUST NOT contain both a "revision-or-derived" extension statement and a "revision-date" statement.
>
> The "revision-or-derived" extension statement MAY be specified multiple times, allowing the import to use any module revision that satisfies at least one of the "revision-or-derived" extension statements.
>
> The "revision-or-derived" extension statement does not guarantee that all module revisions that satisfy an import statement are necessarily compatible; it only gives an indication that the revisions are more likely to be compatible. Hence, NBC changes to an imported module may also require new revisions of any importing modules, updated to accommodation those changes, along with updated import "revision-or-derived" extension statements to depend on the updated imported module revision.
>
> Adding, modifying or removing a "revision-or-derived" extension statement is considered to be a BC change.

4.1. Module import examples

Consider the example module "example-module" from Section 3.5 that is hypothetically available in the following revision/label pairings: 2019-01-01/1.0.0, 2019-02-01/2.0.0, 2019-03-01/3.0.0, 2019-04-01/2.1.0, 2019-05-01/2.2.0 and 2019-06-01/3.1.0. The relationship between the revisions is as before:



4.1.1. Example 1

This example selects module revisions that match, or are derived from the revision 2019-02-01. E.g., this dependency might be used if there was a new container added in revision 2019-02-01 that is augmented by the importing module. It includes revisions/labels: 2019-02-01/2.0.0, 2019-03-01/3.0.0, 2019-04-01/2.1.0, 2019-05-01/2.2.0 and 2019-06-01/3.1.0.

```
import example-module {
  rev:revision-or-derived 2019-02-01;
}
```

Alternatively, the first example could have used the revision label "2.0.0" instead, which selects the same set of revisions/labels.

```
import example-module {
  rev:revision-or-derived 2.0.0;
}
```

4.1.2. Example 2

This example selects module revisions that are derived from 2019-04-01 by using the revision label 2.1.0. It includes revisions/labels: 2019-04-01/2.1.0 and 2019-05-01/2.2.0. Even though 2019-06-01/3.1.0 has a higher revision label number than 2019-04-01/2.1.0 it is not a derived revision, and hence it is not a valid revision for import.

```
import example-module {
  rev:revision-or-derived 2.1.0;
}
```

4.1.3. Example 3

This example selects revisions derived from either 2019-04-01 or 2019-06-01. It includes revisions/labels: 2019-04-01/2.1.0, 2019-05-01/2.2.0, and 2019-06-01/3.1.0.

```
import example-module {
  rev:revision-or-derived 2019-04-01;
  rev:revision-or-derived 2019-06-01;
}
```

5. Updates to ietf-yang-library

This document updates YANG 1.1 [RFC7950] and YANG library [RFC8525] to clarify how ambiguous module imports are resolved. It also defines the YANG module, ietf-yang-library-revisions, that augments YANG library [RFC8525] with revision labels and two leafs to indicate how a server implements deprecated and obsolete schema nodes.

5.1. Resolving ambiguous module imports

A YANG datastore schema, defined in [RFC8525], can specify multiple revisions of a YANG module in the schema using the "import-only" list, with the requirement from [RFC7950] section 5.6.5 that only a single revision of a YANG module may be implemented.

If a YANG module import statement does not specify a specific revision within the datastore schema then it could be ambiguous as to which module revision the import statement should resolve to. Hence, a datastore schema constructed by a client using the information contained in YANG library may not exactly match the datastore schema actually used by the server.

The following two rules remove the ambiguity:

If a module import statement could resolve to more than one module revision defined in the datastore schema, and one of those revisions is implemented (i.e., not an "import-only" module), then the import statement MUST resolve to the revision of the module that is defined as being implemented by the datastore schema.

If a module import statement could resolve to more than one module revision defined in the datastore schema, and none of those revisions are implemented, then the import MUST resolve to the module revision with the latest revision date.

5.2. YANG library versioning augmentations

The "ietf-yang-library-revisions" YANG module has the following structure (using the notation defined in [RFC8340]):

```
module: ietf-yang-library-revisions
  augment /yanglib:yang-library/yanglib:module-set/yanglib:module:
    +--ro revision-label?   rev:revision-label
  augment /yanglib:yang-library/yanglib:module-set/yanglib:module
            /yanglib:submodule:
    +--ro revision-label?   rev:revision-label
  augment /yanglib:yang-library/yanglib:module-set
            /yanglib:import-only-module/yanglib:submodule:
    +--ro revision-label?   rev:revision-label
  augment /yanglib:yang-library/yanglib:schema:
    +--ro deprecated-nodes-implemented?   boolean
    +--ro obsolete-nodes-absent?          boolean
```

5.2.1. Advertising revision-label

The ietf-yang-library-revisions YANG module augments the "module" and "submodule" lists in ietf-yang-library with "revision-label" leafs to optionally declare the revision label associated with each module and submodule.

5.2.2. Reporting how deprecated and obsolete nodes are handled

The ietf-yang-library-revisions YANG module augments YANG library with two boolean leafs to allow a server to report how it implements status "deprecated" and status "obsolete" schema nodes. The leafs are:

deprecated-nodes-implemented: If set to "true", this leaf indicates

that all schema nodes with a status "deprecated" are implemented equivalently as if they had status "current"; otherwise deviations MUST be used to explicitly remove "deprecated" nodes from the schema. If this leaf is set to "false" or absent, then the behavior is unspecified.

obsolete-nodes-absent: If set to "true", this leaf indicates that the server does not implement any status "obsolete" schema nodes. If this leaf is set to "false" or absent, then the behaviour is unspecified.

Servers SHOULD set both the "deprecated-nodes-implemented" and "obsolete-nodes-absent" leafs to "true".

If a server does not set the "deprecated-nodes-implemented" leaf to "true", then clients MUST NOT rely solely on the "rev:non-backwards-compatible-revision" statements to determine whether two module revisions are backwards-compatible, and MUST also consider whether the status of any nodes has changed to "deprecated" and whether those nodes are implemented by the server.

6. Versioning of YANG instance data

Instance data sets [I-D.ietf-netmod-yang-instance-file-format] do not directly make use of the updated revision handling rules described in this document, as compatibility for instance data is undefined.

However, instance data specifies the content-schema of the data-set. This schema SHOULD make use of versioning using revision dates and/or revision labels for the individual YANG modules that comprise the schema or potentially for the entire schema itself (e.g., [I-D.ietf-netmod-yang-packages]).

In this way, the versioning of a content-schema associated with an instance data set may help a client to determine whether the instance data could also be used in conjunction with other revisions of the YANG schema, or other revisions of the modules that define the schema.

7. Guidelines for using the YANG module update rules

The following text updates section 4.7 of [RFC8407] to revise the guidelines for updating YANG modules.

7.1. Guidelines for YANG module authors

All IETF YANG modules MUST include revision-label statements for all newly published YANG modules, and all newly published revisions of existing YANG modules. The revision-label MUST take the form of a YANG semantic version number [I-D.ietf-netmod-yang-semver].

NBC changes to YANG modules may cause problems to clients, who are consumers of YANG models, and hence YANG module authors SHOULD minimize NBC changes and keep changes BC whenever possible.

When NBC changes are introduced, consideration should be given to the impact on clients and YANG module authors SHOULD try to mitigate that impact.

A "rev:non-backwards-compatible-revision" statement MUST be added if there are NBC changes relative to the previous revision.

Removing old revision statements from a module's revision history could break import by revision, and hence it is RECOMMENDED to retain them. If all dependencies have been updated to not import specific revisions of a module, then the corresponding revision statements can be removed from that module. An alternative solution, if the revision section is too long, would be to remove, or curtail, the older description statements associated with the previous revisions.

The "rev:revision-or-derived" extension SHOULD be used in YANG module imports to indicate revision dependencies between modules in preference to the "revision-date" statement, which causes overly strict import dependencies and SHOULD NOT be used.

A module that includes submodules SHOULD use the "revision-date" statement to include specific submodule revisions. The revision of the including module MUST be updated when any included submodule has changed.

In some cases a module or submodule revision that is not strictly NBC by the definition in Section 3.1.2 of this specification may include the "non-backwards-compatible" statement. Here is an example when adding the statement may be desirable:

* A "config false" leaf had its value space expanded (for example, a range was increased, or additional enum values were added) and the author or server implementor feels there is a significant compatibility impact for clients and users of the module or submodule

7.1.1. Making non-backwards-compatible changes to a YANG module

There are various valid situations where a YANG module has to be modified in an NBC way. Here are some guidelines on how non-backwards-compatible changes can be made incrementally, with the assumption that deprecated nodes are implemented by the server, and obsolete nodes are not:

1. The changes should be made gradually, e.g., a data node's status SHOULD NOT be changed directly from "current" to "obsolete" (see Section 4.7 of [RFC8407]), instead the status SHOULD first be marked "deprecated". At some point in the future, when support is removed for the data node, there are two options. The first, and preferred, option is to keep the data node definition in the model and change the status to "obsolete". The second option is to simply remove the data node from the model, but this has the risk of breaking modules which import the modified module, and the removed identifier may be accidently reused in a future revision.

2. For deprecated data nodes the "description" statement SHOULD also indicate until when support for the node is guaranteed (if known). If there is a replacement data node, rpc, action or notification for the deprecated node, this SHOULD be stated in the "description". The reason for deprecating the node can also be included in the "description" if it is deemed to be of potential interest to the user.

3. For obsolete data nodes, it is RECOMMENDED to keep the above information, from when the node had status "deprecated", which is still relevant.

4. When obsoleting or deprecating data nodes, the "deprecated" or "obsolete" status SHOULD be applied at the highest possible level in the data tree. For clarity, the "status" statement SHOULD also be applied to all descendent data nodes, but the additional status related information does not need to be repeated if it does not introduce any additional information.

5. NBC changes which can break imports SHOULD be avoided because of the impact on the importing module. The importing modules could get broken, e.g., if an augmented node in the importing module has been removed from the imported module. Alternatively, the schema of the importing modules could undergo an NBC change due to the NBC change in the imported module, e.g., if a node in a grouping has been removed. As described in Appendix B.1, instead of removing a node, that node SHOULD first be deprecated and then obsoleted.

See Appendix B for examples on how NBC changes can be made.

7.2. Versioning Considerations for Clients

Guidelines for clients of modules using the new module revision update procedure:

* Clients SHOULD be liberal when processing data received from a server. For example, the server may have increased the range of an operational node causing the client to receive a value which is outside the range of the YANG model revision it was coded against.

* Clients SHOULD monitor changes to published YANG modules through their revision history, and use appropriate tooling to understand the specific changes between module revision. In particular, clients SHOULD NOT migrate to NBC revisions of a module without understanding any potential impact of the specific NBC changes.

* Clients SHOULD plan to make changes to match published status changes. When a node's status changes from "current" to "deprecated", clients SHOULD plan to stop using that node in a timely fashion. When a node's status changes to "obsolete", clients MUST stop using that node.

8. Module Versioning Extension YANG Modules

YANG module with extension statements for annotating NBC changes, revision label, revision label scheme, and importing by revision.

<CODE BEGINS> file "ietf-yang-revisions@2022-08-22.yang"

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

  // RFC Ed.: We need the bis version to get the new type revision-identifier
  // If 6991-bis is not yet an RFC we need to copy the definition here
  import ietf-yang-types {
    prefix yang;
    reference
      "XXXX [ietf-netmod-rfc6991-bis]: Common YANG Data Types";
  }

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

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

       Author:   Reshad Rahman
                 <mailto:reshad@yahoo.com>

       Author:   Robert Wilton
                 <mailto:rwilton@cisco.com>

       Author:   Balazs Lengyel
                 <mailto:balazs.lengyel@ericsson.com>

       Author:   Jason Sterne
                 <mailto:jason.sterne@nokia.com>";
     description
       "This YANG 1.1 module contains definitions and extensions to
       support updated YANG revision handling.

       Copyright (c) 2021 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.

       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.";

     // RFC Ed.: update the date below with the date of RFC publication
     // and remove this note.
     // RFC Ed.: replace XXXX (inc above) with actual RFC number and
     // remove this note.

     revision 2022-10-10 {
       rev:revision-label 1.0.0-draft-ietf-netmod-yang-module-versioning-07;
       description
         "Initial version.";
       reference
         "XXXX: Updated YANG Module Revision Handling";
```

```
  }

  typedef revision-label {
    type string {
      length "1..255";
      pattern '[a-zA-Z0-9,\-_.+]+';
      pattern '\d{4}-\d{2}-\d{2}' {
        modifier invert-match;
      }
    }
    description
      "A label associated with a YANG revision.

      Alphanumeric characters, comma, hyphen, underscore, period
      and plus are the only accepted characters. MUST NOT match
      revision-date.";
    reference
      "XXXX: Updated YANG Module Revision Handling;
      Section 3.3, Revision label";
  }

  typedef revision-date-or-label {
    type union {
      type yang:revision-identifier;
      type revision-label;
    }
    description
      "Represents either a YANG revision date or a revision label";
  }

  extension non-backwards-compatible-revision {
    description
      "This statement is used to indicate YANG module revisions that
      contain non-backwards-compatible changes.

      The statement MUST only be a substatement of the 'revision'
      statement.  Zero or one 'non-backwards-compatible' statements
      per parent statement is allowed.  No substatements for this
      extension have been standardized.

      If a revision of a YANG module contains changes, relative to
      the preceding revision in the revision history, that do not
      conform to the backwards-compatible module update rules defined
      in RFC-XXX, then the 'non-backwards-compatible' statement MUST
      be added as a substatement to the revision statement.

      Conversely, if a revision does not contain a
      'non-backwards-compatible' statement then all changes,
```

```
        relative to the preceding revision in the revision history,
        MUST be backwards-compatible.

        A new module revision that only contains changes that are
        backwards-compatible SHOULD NOT include the
        'non-backwards-compatible' statement.  An example of when
        an author might add the 'non-backwards-compatible' statement
        is if they believe a change could negatively impact clients
        even though the backwards compatibility rules defined in
        RFC-XXXX classify it as a backwards-compatible change.

        Add, removing, or changing a 'non-backwards-compatible'
        statement is a backwards-compatible version change.";

      reference
        "XXXX: Updated YANG Module Revision Handling;
        Section 3.2, non-backwards-compatible revision extension statement";
    }

    extension revision-label {
      argument revision-label;
      description
        "The revision label can be used to provide an additional
        versioning identifier associated with a module or submodule
        revision.  One such scheme that
        could be used is [XXXX: ietf-netmod-yang-semver].

        The format of the revision-label argument MUST conform to the
        pattern defined for the revision-label typedef in this module.

        The statement MUST only be a substatement of the revision
        statement.  Zero or one revision-label statements per parent
        statement are allowed.  No substatements for this extension
        have been standardized.

        Revision labels MUST be unique amongst all revisions of a
        module or submodule.

        Adding a revision label is a backwards-compatible version
        change.  Changing or removing an existing revision label in
        the revision history is a non-backwards-compatible version
        change, because it could impact any references to that
        revision label.";

      reference
        "XXXX: Updated YANG Module Revision Handling;
        Section 3.3, Revision label";
    }
```

```
extension revision-label-scheme {
  argument revision-label-scheme-base;
  description
    "The revision label scheme specifies which revision-label scheme
    the module or submodule uses.

    The mandatory revision-label-scheme-base argument MUST be an
    identity derived from revision-label-scheme-base.

    This extension is only valid as a top-level statement, i.e.,
    given as as a substatement to 'module' or 'submodule'.  No
    substatements for this extension have been standardized.

    This extension MUST be used if there is a revision-label
    statement in the module or submodule.

    Adding a revision label scheme is a backwards-compatible version
    change.  Changing a revision label scheme is a
    non-backwards-compatible version change, unless the new revision
    label scheme is backwards-compatible with the replaced revision
    label scheme.  Removing a revision label scheme is a
    non-backwards-compatible version change.";

  reference
    "XXXX: Updated YANG Module Revision Handling;
    Section 3.3.1, Revision label scheme extension statement";
}

extension revision-or-derived {
  argument revision-date-or-label;
  description
    "Restricts the revision of the module that may be imported to
    one that matches or is derived from the specified
    revision-date or revision-label.

    The argument value MUST conform to the
    'revision-date-or-label' defined type.

    The statement MUST only be a substatement of the import
    statement.  Zero, one or more 'revision-or-derived' statements
    per parent statement are allowed.  No substatements for this
    extension have been standardized.

    If specified multiple times, then any module revision that
    satisfies at least one of the 'revision-or-derived' statements
    is an acceptable revision for import.

    An 'import' statement MUST NOT contain both a
```

```
        'revision-or-derived' extension statement and a
        'revision-date' statement.

        A particular revision of an imported module satisfies an
        import's 'revision-or-derived' extension statement if the
        imported module's revision history contains a revision
        statement with a matching revision date or revision label.

        The 'revision-or-derived' extension statement does not
        guarantee that all module revisions that satisfy an import
        statement are necessarily compatible, it only gives an
        indication that the revisions are more likely to be
        compatible.

        Adding, removing or updating a 'revision-or-derived'
        statement to an import is a backwards-compatible change.
        ";

      reference
        "XXXX: Updated YANG Module Revision Handling;
        Section 4, Import by derived revision";
    }

    identity revision-label-scheme-base {
      description
        "Base identity from which all revision label schemes are
        derived.";

        reference
          "XXXX: Updated YANG Module Revision Handling;
          Section 3.3.1, Revision label scheme extension statement";

    }




}
<CODE ENDS>

YANG module with augmentations to YANG Library to revision labels

<CODE BEGINS> file "ietf-yang-library-revisions@2021-11-04.yang"
module ietf-yang-library-revisions {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-library-revisions";
  prefix yl-rev;
```

```
  import ietf-yang-revisions {
    prefix rev;
    reference
      "XXXX: Updated YANG Module Revision Handling";
  }

  import ietf-yang-library {
    prefix yanglib;
    reference "RFC 8525: YANG Library";
  }

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

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

     Author:   Reshad Rahman
               <mailto:reshad@yahoo.com>

     Author:   Robert Wilton
               <mailto:rwilton@cisco.com>

     Author:   Balazs Lengyel
               <mailto:balazs.lengyel@ericsson.com>

     Author:   Jason Sterne
               <mailto:jason.sterne@nokia.com>";
  description
    "This module contains augmentations to YANG Library to add module
     level revision label and to provide an indication of how
     deprecated and obsolete nodes are handled by the server.

     Copyright (c) 2021 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.
```

```
        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.";

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    // RFC Ed.: replace XXXX (including in the imports above) with
    // actual RFC number and remove this note.
    // RFC Ed.: please replace revision-label version with 1.0.0 and
    // remove this note.
    revision 2021-11-04 {
      rev:revision-label 1.0.0-draft-ietf-netmod-yang-module-versioning-05;
      description
        "Initial revision";
      reference
        "XXXX: Updated YANG Module Revision Handling";
    }

    // library 1.0 modules-state is not augmented with revision-label

    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
      description
        "Add a revision label to module information";
      leaf revision-label {
        type rev:revision-label;
        description
          "The revision label associated with this module revision.
           The label MUST match the rev:revision-label value in the specific
           revision of the module loaded in this module-set.";

        reference
          "XXXX: Updated YANG Module Revision Handling;
           Section 5.2.1, Advertising revision-label";
      }
    }

    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module/"
            + "yanglib:submodule" {
      description
        "Add a revision label to submodule information";
      leaf revision-label {
        type rev:revision-label;
        description
          "The revision label associated with this submodule revision.
           The label MUST match the rev:revision-label value in the specific
           revision of the submodule included by the module loaded in
```

```
            this module-set.";

        reference
          "XXXX: Updated YANG Module Revision Handling;
           Section 5.2.1, Advertising revision-label";
      }
    }

    augment "/yanglib:yang-library/yanglib:module-set/"
            + "yanglib:import-only-module" {
      description
        "Add a revision label to module information";
      leaf revision-label {
        type rev:revision-label;
        description
          "The revision label associated with this module revision.
           The label MUST match the rev:revision-label value in the specific
           revision of the module included in this module-set.";

        reference
          "XXXX: Updated YANG Module Revision Handling;
           Section 5.2.1, Advertising revision-label";
      }
    }

    augment "/yanglib:yang-library/yanglib:module-set/"
            + "yanglib:import-only-module/yanglib:submodule" {
      description
        "Add a revision label to submodule information";
      leaf revision-label {
        type rev:revision-label;
        description
          "The revision label associated with this submodule revision.
           The label MUST match the rev:label value in the specific
           revision of the submodule included by the
           import-only-module loaded in this module-set.";

        reference
          "XXXX: Updated YANG Module Revision Handling;
           Section 5.2.1, Advertising revision-label";
      }
    }

    augment "/yanglib:yang-library/yanglib:schema" {
      description
        "Augmentations to the ietf-yang-library module to indicate how
         deprecated and obsoleted nodes are handled for each datastore
         schema supported by the server.";
```

```
      leaf deprecated-nodes-implemented {
        type boolean;
        description
          "If set to true, this leaf indicates that all schema nodes with
           a status 'deprecated' are implemented
           equivalently as if they had status 'current'; otherwise
           deviations MUST be used to explicitly remove deprecated
           nodes from the schema.  If this leaf is absent or set to false,
           then the behavior is unspecified.";

        reference
          "XXXX: Updated YANG Module Revision Handling;
           Section 5.2.2, Reporting how deprecated and obsolete nodes
           are handled";
      }

      leaf obsolete-nodes-absent {
        type boolean;
        description
          "If set to true, this leaf indicates that the server does not
           implement any status 'obsolete' schema nodes.  If this leaf is
           absent or set to false, then the behaviour is unspecified.";

        reference
          "XXXX: Updated YANG Module Revision Handling;
           Section 5.2.2, Reporting how deprecated and obsolete nodes
           are handled";
      }
    }
  }
  <CODE ENDS>
```

9. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The following individuals are (or have been) members of the design team and have worked on the YANG versioning project:

* Balazs Lengyel
* Benoit Claise
* Bo Wu
* Ebben Aries
* Jan Lindblad

* Jason Sterne

* Joe Clarke

* Juergen Schoenwaelder

* Mahesh Jethanandani

* Michael (Wangzitao)

* Qin Wu

* Reshad Rahman

* Rob Wilton

The initial revision of this document was refactored and built upon [I-D.clacla-netmod-yang-model-update]. We would like to thank Kevin D'Souza and Benoit Claise for their initial work in this problem space.

Discussons on the use of Semver for YANG versioning has been held with authors of the OpenConfig YANG models. We would like to thank both Anees Shaikh and Rob Shakir for their input into this problem space.

We would also like to thank Lou Berger, Andy Bierman, Martin Bjorklund, Italo Busi, Tom Hill, Scott Mansfield, Kent Watsen for their contributions and review comments.

10. Security Considerations

The document does not define any new protocol or data model. There are no security considerations beyond those specified in [RFC7950] and [RFC6020].

11. IANA Considerations

11.1. YANG Module Registrations

This document requests IANA to registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registrations are requested.

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

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

The following YANG module is requested to be registred in the "IANA Module Names" [RFC6020]. Following the format in RFC 6020, the following registrations are requested:

The ietf-yang-revisions module:

Name: ietf-yang-revisions

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-revisions

Prefix: rev

Reference: [RFCXXXX]

The ietf-yang-library-revisions module:

Name: ietf-yang-library-revisions

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-library-revisions

Prefix: yl-rev

Reference: [RFCXXXX]

11.2. Guidance for versioning in IANA maintained YANG modules

Note for IANA (to be removed by the RFC editor): Please check that the registries and IANA YANG modules are referenced in the appropriate way.

IANA is responsible for maintaining and versioning YANG modules that are derived from other IANA registries. For example, "iana-if-type.yang" [IfTypeYang] is derived from the "Interface Types (ifType) IANA registry" [IfTypesReg], and "iana-routing-types.yang" [RoutingTypesYang] is derived from the "Address Family Numbers" [AddrFamilyReg] and "Subsequent Address Family Identifiers (SAFI) Parameters" [SAFIReg] IANA registries.

Normally, updates to the registries cause any derived YANG modules to be updated in a backwards-compatible way, but there are some cases where the registry updates can cause non-backward-compatible updates to the derived YANG module. An example of such an update is the 2020-12-31 revision of iana-routing-types.yang [RoutingTypesDecRevision], where the enum name for two SAFI values was changed.

In all cases, IANA MUST follow the versioning guidance specified in Section 3.1, and MUST include a "rev:non-backwards-compatible-revision" substatement to the latest revision statement whenever an IANA maintained module is updated in a non-backwards-compatible way, as described in Section 3.2.

Note: For published IANA maintained YANG modules that contain non-backwards-compatible changes between revisions, a new revision should be published with the "rev:non-backwards-compatible-revision" substatement retrospectively added to any revisions containing non-backwards-compatible changes.

Non-normative examples of updates to enumeration types in IANA maintained modules that would be classified as non-backwards-compatible changes are: Changing the status of an enumeration typedef to obsolete, changing the status of an enum entry to obsolete, removing an enum entry, changing the identifier of an enum entry, or changing the described meaning of an enum entry.

Non-normative examples of updates to enumeration types in IANA maintained modules that would be classified as backwards-compatible changes are: Adding a new enum entry to the end of the enumeration, changing the status or an enum entry to deprecated, or improving the description of an enumeration that does not change its defined meaning.

Non-normative examples of updates to identity types in IANA maintained modules that would be classified as non-backwards-compatible changes are: Changing the status of an identity to obsolete, removing an identity, renaming an identity, or changing the described meaning of an identity.

Non-normative examples of updates to identity types in IANA maintained modules that would be classified as backwards-compatible changes are: Adding a new identity, changing the status or an identity to deprecated, or improving the description of an identity that does not change its defined meaning.

12. References

12.1. Normative References

[I-D.ietf-netmod-rfc6991-bis]
Schoenwaelder, J., "Common YANG Data Types", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc6991-bis-13, 22 March 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-rfc6991-bis-13.txt>.

[I-D.ietf-netmod-yang-semver]
Clarke, J., Wilton, R., Rahman, R., Lengyel, B., Sterne, J., and B. Claise, "YANG Semantic Versioning", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-semver-07, 10 July 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-semver-07.txt>.

[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>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <https://www.rfc-editor.org/info/rfc8525>.

12.2. Informative References

[AddrFamilyReg]
"Address Family Numbers IANA Registry", <https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml>.

[I-D.clacla-netmod-yang-model-update]
Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <https://www.ietf.org/archive/id/draft-clacla-netmod-yang-model-update-06.txt>.

[I-D.ietf-netmod-yang-instance-file-format]
Lengyel, B. and B. Claise, "YANG Instance Data File Format", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-instance-file-format-21, 8 October 2021, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-instance-file-format-21.txt>.

[I-D.ietf-netmod-yang-packages]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Packages", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-packages-03, 4 March 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-packages-03.txt>.

[I-D.ietf-netmod-yang-schema-comparison]
Wilton, R., "YANG Schema Comparison", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-schema-comparison-01, 2 November 2020, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-schema-comparison-01.txt>.

[I-D.ietf-netmod-yang-solutions]
Wilton, R., "YANG Versioning Solution Overview", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-solutions-01, 2 November 2020, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-solutions-01.txt>.

[I-D.ietf-netmod-yang-ver-selection]
Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Schema Selection", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-ver-selection-00, 17 March 2020, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-ver-selection-00.txt>.

[I-D.ietf-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-versioning-reqs-07, 10 July 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-versioning-reqs-07.txt>.

[IfTypesReg]
"Interface Types (ifType) IANA Registry", <https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#smi-numbers-5>.

[IfTypeYang]
"iana-if-type YANG Module", <https://www.iana.org/assignments/iana-if-type/iana-if-type.xhtml>.

[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>.

[RoutingTypesDecRevision]
"2020-12-31 revision of iana-routing-types.yang", <https://www.iana.org/assignments/yang-parameters/iana-routing-types@2020-12-31.yang>.

[RoutingTypesYang]
"iana-routing-types YANG Module", <https://www.iana.org/assignments/iana-routing-types/iana-routing-types.xhtml>.

[SAFIReg] "Subsequent Address Family Identifiers (SAFI) Parameters IANA Registry", <https://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml>.

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

Appendix A. Examples of changes that are NBC

Examples of NBC changes include:

* Deleting a data node, or changing it to status obsolete.

* Changing the name, type, or units of a data node.

* Modifying the description in a way that changes the semantic meaning of the data node.

* Any changes that remove any previously allowed values from the allowed value set of the data node, either through changes in the type definition, or the addition or changes to "must" statements, or changes in the description.

* Adding or modifying "when" statements that reduce when the data node is available in the schema.

* Making the statement conditional on if-feature.

Appendix B. Examples of applying the NBC change guidelines

The following sections give steps that could be taken for making NBC changes to a YANG module or submodule using the incremental approach described in section Section 7.1.1.

The examples are all for "config true" nodes.

B.1. Removing a data node

Removing a leaf or container from the data tree, e.g., because support for the corresponding feature is being removed:

1. The schema node's status is changed to "deprecated" and the node is supported for some period of time (e.g. one year). This is a BC change.

2. When the schema node is not supported anymore, its status is changed to "obsolete" and the "description" updated. This is an NBC change.

B.2. Changing the type of a leaf node

Changing the type of a leaf node. e.g., a "vpn-id" node of type integer being changed to a string:

1. The status of schema node "vpn-id" is changed to "deprecated" and the node is supported for some period of time (e.g. one year). This is a BC change. The description is updated to indicate that "vpn-name" is replacing this node.

2. A new schema node, e.g., "vpn-name", of type string is added to the same location as the existing node "vpn-id". This new node has status "current" and its description explains that it is replacing node "vpn-id".

3. During the period of time when both schema nodes are supported, the interactions between the two nodes is outside the scope of this document and will vary on a case by case basis. One possible option is to have the server prevent the new node from being set if the old node is already set (and vice-versa). The new node could have a "when" statement added to it to achieve this. The old node, however, must not have a "when" statement added, or an existing "when" modified to be more restrictive, since this would be an NBC change. In any case, the server could reject the old node from being set if the new node is already set.

4. When the schema node "vpn-id" is not supported anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.

B.3. Reducing the range of a leaf node

Reducing the range of values of a leaf-node, e.g., consider a "vpn-id" schema node of type uint32 being changed from range 1..5000 to range 1..2000:

1. If all values which are being removed were never supported, e.g., if a vpn-id of 2001 or higher was never accepted, this is a BC change for the functionality (no functionality change). Even if it is an NBC change for the YANG model, there should be no impact for clients using that YANG model.

2. If one or more values being removed was previously supported, e.g., if a vpn-id of 3333 was accepted previously, this is an NBC change for the YANG model. Clients using the old YANG model will be impacted, so a change of this nature should be done carefully, e.g., by using the steps described in Appendix B.2

B.4. Changing the key of a list

Changing the key of a list has a big impact to the client. For example, consider a "sessions" list which has a key "interface" and there is a need to change the key to "dest-address". Such a change can be done in steps:

1. The status of list "sessions" is changed to "deprecated" and the list is supported for some period of time (e.g. one year). This is a BC change. The description is updated to indicate the new list that is replacing this list.

2. A new list is created in the same location with the same descendant schema nodes but with "dest-address" as key. Finding an appropriate name for the new list can be difficult. In this case the new list is called "sessions-address", has status "current" and its description should explain that it is replacing list "session".

3. During the period of time when both lists are supported, the interactions between the two lists is outside the scope of this document and will vary on a case by case basis. One possible option is to have the server prevent entries in the new list from being created if the old list already has entries (and vice-versa).

4. When list "sessions" is not available anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.

B.5. Renaming a node

A leaf or container schema node may be renamed, either due to a spelling error in the previous name or because of a better name. For example a node "ip-adress" could be renamed to "ip-address":

1. The status of the existing node "ip-adress" is changed to "deprecated" and is supported for some period of time (e.g. one year). This is a BC change. The description is updated to indicate the node that is replacing this node.

2. The new schema node "ip-address" is added to the same location as the existing node "ip-adress". This new node has status "current" and its description should explain that it is replacing node "ip-adress".

3. During the period of time when both nodes are available, the interactions between the two nodes is outside the scope of this document and will vary on a case by case basis. One possible option is to have the server prevent the new node from being set if the old node is already set (and vice-versa). The new node could have a "when" statement added to it to achieve this. The old node, however, must not have a "when" statement added, or an existing "when" modified to be more restrictive, since this would be an NBC change. In any case, the server could reject the old node from being set if the new node is already set.

4. When node "ip-adress" is not available anymore, its status is changed to "obsolete" and the "description" is updated. This is an NBC change.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems, Inc.
Email: rwilton@cisco.com

Reshad Rahman (editor)
Email: reshad@yahoo.com

Balazs Lengyel (editor)
Ericsson
Email: balazs.lengyel@ericsson.com

Joe Clarke
Cisco Systems, Inc.
Email: jclarke@cisco.com

Jason Sterne
Nokia
Email: jason.sterne@nokia.com

Network Working Group
Internet-Draft
Updates: 8407 (if approved)
Intended status: Standards Track
Expires: 27 April 2023

J. Clarke, Ed.
R. Wilton, Ed.
Cisco Systems, Inc.
R. Rahman

B. Lengyel
Ericsson
J. Sterne
Nokia
B. Claise
Huawei

24 October 2022

YANG Semantic Versioning
draft-ietf-netmod-yang-semver-08

Abstract

This document specifies a scheme and guidelines for applying an extended set of semantic versioning rules to revisions of YANG artifacts (e.g., modules and packages). Additionally, this document defines an RFCAAAA-compliant revision-label-scheme for this YANG semantic versioning scheme.

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 27 April 2023.

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-netmod-yang-module-versioning] puts forth a number of concepts relating to modified rules for updating YANG modules and submodules, a means to signal when a new revision of a module or submodule has non-backwards-compatible (NBC) changes compared to its previous revision, and a scheme that uses the revision history as a lineage for determining from where a specific revision of a YANG module or submodule is derived. Additionally, section 3.4 of [I-D.ietf-netmod-yang-module-versioning] defines a revision-label which can be used as an alias to provide additional context or as a meaningful label to refer to a specific revision.

This document defines a revision-label scheme that uses extended semantic versioning rules [SemVer] for YANG artifacts (i.e., YANG modules, YANG submodules, and YANG packages [I-D.ietf-netmod-yang-packages]) as well as the revision label definition for using this scheme. The goal being to add a human readable revision label that provides compatibility information for the YANG artifact without needing to compare or parse its body. The label and rules defined herein represent the RECOMMENDED revision label scheme for IETF YANG artifacts.

Note that a specific revision of the SemVer 2.0.0 specification is referenced here (from June 19, 2020) to provide an immutable version. This is because the 2.0.0 version of the specification has changed over time without any change to the semantic version itself. In some cases the text has changed in non-backwards-compatible ways.

2. Terminology and Conventions

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.

Additionally, this document uses the following terminology:

* YANG artifact: YANG modules, YANG submodules, and YANG packages [I-D.ietf-netmod-yang-packages] are examples of YANG artifacts for the purposes of this document.

* SemVer: A version string that corresponds to the rules defined in [SemVer]. This specific camel-case notation is the one used by the SemVer 2.0.0 website and used within this document to distinguish between YANG Semver.

* YANG Semver: A revision-label identifier that is consistent with the extended set of semantic versioning rules, based on [SemVer] , defined within this document.

3. YANG Semantic Versioning

This section defines YANG Semantic Versioning, explains how it is used with YANG artifacts, and describes the rules associated with changing an artifact's semantic version when its contents are updated.

3.1. Relationship Between SemVer and YANG Semver

[SemVer] is completely compatible with YANG Semver in that a SemVer semantic version number is legal according to the YANG Semver rules (though the inverse is not necessarily true). YANG Semver is a superset of the SemVer rules, and allow for limited branching within YANG artifacts. If no branching occurs within a YANG artifact (i.e., you do not use the compatibility modifiers described below), the YANG Semver version label will appear as a SemVer version number.

3.2. YANG Semver Pattern

YANG artifacts that employ semantic versioning as defined in this document MUST use a version string (e.g., in revision-label or as a package version) that corresponds to the following pattern: 'X.Y.Z_COMPAT'. Where:

* X, Y and Z are mandatory non-negative integers that are each less than or equal to 2147483647 (i.e., the maximum signed 32-bit integer value) and MUST NOT contain leading zeroes,

* The '.' is a literal period (ASCII character 0x2e),

* The '_' is an optional single literal underscore (ASCII character 0x5f) and MUST only be present if the following COMPAT element is included,

* COMPAT, if specified, MUST be either the literal string "compatible" or the literal string "non_compatible".

Additionally, [SemVer] defines two specific types of metadata that may be appended to a semantic version string. Pre-release metadata MAY be appended to a YANG Semver string after a trailing '-' character. Build metadata MAY be appended after a trailing '+' character. If both pre-release and build metadata are present, then build metadata MUST follow pre-release metadata. While build metadata MUST be ignored when comparing YANG semantic versions, pre-

release metadata MUST be used during module and submodule development as specified in Section 5 . Both pre-release and build metadata are allowed in order to support all the [SemVer] rules. Thus, a version lineage that follows strict [SemVer] rules is allowed for a YANG artifact.

To signal the use of this versioning scheme, modules and submodules MUST set the revision-label-scheme extension, as defined in [I-D.ietf-netmod-yang-module-versioning] , to the identity "yang-semver". That identity value is defined in the ietf-yang-semver module below.

Additionally, this ietf-yang-semver module defines a typedef that formally specifies the syntax of the YANG Semver.

3.3. Semantic Versioning Scheme for YANG Artifacts

This document defines the YANG semantic versioning scheme that is used for YANG artifacts that employ the YANG Semver label. The versioning scheme has the following properties:

* The YANG semantic versioning scheme is extended from version 2.0.0 of the semantic versioning scheme defined at semver.org [SemVer] to cover the additional requirements for the management of YANG artifact lifecyles that cannot be addressed using the semver.org 2.0.0 versioning scheme alone.

* Unlike the [SemVer] versioning scheme, the YANG semantic versioning scheme supports updates to older versions of YANG artifacts, to allow for bug fixes and enhancements to artifact versions that are not the latest. However, it does not provide for the unlimited branching and updating of older revisions which are documented by the general rules in [I-D.ietf-netmod-yang-module-versioning] .

* YANG artifacts that follow the [SemVer] versioning scheme are fully compatible with implementations that understand the YANG semantic versioning scheme defined in this document.

* If updates are always restricted to the latest revision of the artifact only, then the version numbers used by the YANG semantic versioning scheme are exactly the same as those defined by the [SemVer] versioning scheme.

Every YANG module and submodule versioned using the YANG semantic versioning scheme specifies the module’s or submodule’s semantic version as the argument to the ’rev:revision-label’ statement.

Because the rules put forth in [I-D.ietf-netmod-yang-module-versioning] are designed to work well with existing versions of YANG and allow for artifact authors to migrate to this scheme, it is not expected that all revisions of a given YANG artifact will have a semantic version label. For example, the first revision of a module or submodule may have been produced before this scheme was available.

YANG packages that make use of this YANG Semver will reflect that in the package metadata.

As stated above, the YANG semantic version is expressed as a string of the form: 'X.Y.Z_COMPAT'.

* 'X' is the MAJOR version. Changes in the MAJOR version number indicate changes that are non-backwards-compatible to versions with a lower MAJOR version number.

* 'Y' is the MINOR version. Changes in the MINOR version number indicate changes that are backwards-compatible to versions with the same MAJOR version number, but a lower MINOR version number and no "_compatible" or "_non_compatible" modifier.

* 'Z' is the PATCH version. Changes in the PATCH version number can indicate an editorial change to the YANG artifact. In conjunction with the '_COMPAT' modifier (see below) changes to 'Z' may indicate a more substantive module change. An editorial change is defined to be a change in the YANG artifact's content that does not affect the semantic meaning or functionality provided by the artifact in any way. Some examples include correcting a spelling mistake in the description of a leaf within a YANG module or submodule, non-significant whitespace changes (e.g., realigning description statements or changing indentation), or changes to YANG comments. Note: restructuring how a module uses, or does not use, submodules is treated as an editorial level change on the condition that there is no change in the module's semantic behavior due to the restructuring.

* '_COMPAT' is an additional modifier, unique to YANG Semver (i.e., not valid in [SemVer]), that indicates backwards-compatible, or non-backwards-compatible changes relative to versions with the same MAJOR and MINOR version numbers, but lower PATCH version number, depending on what form modifier '_COMPAT' takes:

  - If the modifier string is absent, the change represents an editorial change.

- If, however, the modifier string is present, the meaning is described below:

- "_compatible" - the change represents a backwards-compatible change

- "_non_compatible" - the change represents a non-backwards-compatible change

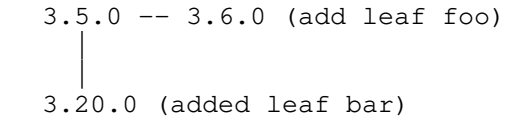
The '_COMPAT' modifier string is "sticky". Once a revision of a module has a modifier in the revision label, then all descendants of that revision with the same X.Y version digits will also have a modifier. The modifier can change from "_compatible" to "_non_compatible" in a descendant revision, but the modifier MUST NOT change from "_non_compatible" to "_compatible" and MUST NOT be removed. The persistence of the "_non_compatible" modifier ensures that comparisons of revision labels do not give the false impression of compatibility between two potentially non-compatible revisions. If "_non_compatible" was removed, for example between revisions "3.3.2_non_compatible" and "3.3.3" (where "3.3.3" was simply an editorial change), then comparing revision labels of "3.3.3" back to an ancestor "3.0.0" would look like they are backwards compatible when they are not (since "3.3.2_non_compatible" was in the chain of ancestors and introduced a non-backwards-compatible change).

The YANG artifact name and YANG semantic version uniquely identify a revision of said artifact. There MUST NOT be multiple instances of a YANG artifact definition with the same name and YANG semantic version but different content (and in the case of modules and submodules, different revision dates).

There MUST NOT be multiple versions of a YANG artifact that have the same MAJOR, MINOR and PATCH version numbers, but different patch modifier strings. E.g., artifact version "1.2.3_non_compatible" MUST NOT be defined if artifact version "1.2.3" has already been defined.

3.3.1. Branching Limitations with YANG Semver

YANG artifacts that use the YANG Semver revision-label scheme MUST ensure that two artifacts with the same MAJOR version number and no _compatible or _non_compatible modifiers are backwards compatible. Therefore, certain branching schemes cannot be used with YANG Semver. For example, the following branched parent-child module relationship using the following YANG Semver revision labels is not supported:



In this case, given only the revision labels 3.6.0 and 3.20.0 without any parent-child relationship information, one would assume that 3.20.0 is backwards compatible with 3.6.0. But in the illegal example above, 3.20.0 is not backwards compatible with 3.6.0 since 3.20.0 does not contain the leaf foo.

Note that this type of branched parent-child relationship, where two revisions have different backwards compatible changes based on the same parent, is allowed in [I-D.ietf-netmod-yang-module-versioning].

3.3.2. YANG Semver with submodules

YANG Semver MAY be used to version submodules. Submodule version are separate of any version on the including module, but if a submodule has changed, then the version of the including module MUST also be updated.

The rules for determining the version change of a submodule are the same as those defined in Section 3.2 and Section 3.3 as applied to YANG modules, except they only apply to the part of the module schema defined within the submodule's file.

One interesting case is moving definitions from one submodule to another in a way that does not change the resultant schema of the including module. In this case:

1. The including module has editorial changes

2. The submodule with the schema definition removed has non-backwards-compatible changes

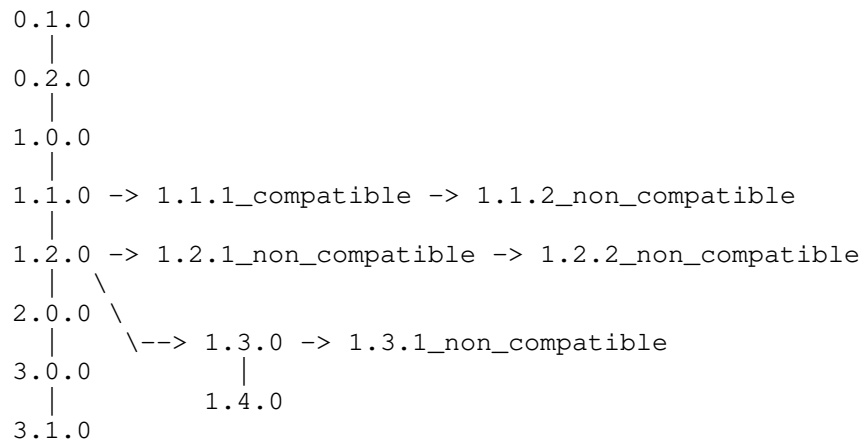
3. The submodule with the schema definitions added has backwards-compatible changes

Note that the meaning of a submodule may change drastically despite having no changes in content or revision due to changes in other submodules belonging to the same module (e.g. groupings and typedefs declared in one submodule and used in another).

3.3.3. Examples for YANG semantic versions

The following diagram and explanation illustrate how YANG semantic versions work.

YANG Semantic versions for an example module:



The tree diagram above illustrates how the version history might evolve for an example module. The tree diagram only shows the parent/child ancestry relationships between the revisions. It does not describe the chronology of the revisions (i.e. when in time each revision was published relative to the other revisions).

The following description lists an example of what the chronological order of the revisions could look like, from oldest revision to newest:

0.1.0 - first pre-release module version

0.2.0 - second pre-release module version (with NBC changes)

1.0.0 - first release (may have NBC changes from 0.2.0)

1.1.0 - added new functionality, leaf "foo" (BC)

1.2.0 - added new functionality, leaf "baz" (BC)

2.0.0 - change existing model for performance reasons, e.g. re-key list (NBC)

1.3.0 - improve existing functionality, added leaf "foo-64" (BC)

1.1.1_compatible - backport "foo-64" leaf to 1.1.x to avoid implementing "baz" from 1.2.0. This revision was created after 1.2.0 otherwise it may have been released as 1.2.0. (BC)

3.0.0 - NBC bugfix, rename "baz" to "bar"; also add new BC leaf "wibble"; (NBC)

1.3.1_non_compatible - backport NBC fix, rename "baz" to "bar" (NBC)

1.2.1_non_compatible - backport NBC fix, rename "baz" to "bar" (NBC)

1.1.2_non_compatible - NBC point bug fix, not required in 2.0.0 due to model changes (NBC)

1.4.0 - introduce new leaf "ghoti" (BC)

3.1.0 - introduce new leaf "wobble" (BC)

1.2.2_non_compatible - backport "wibble". This is a BC change but "non_compatible" modifier is sticky. (BC)

The partial ancestry relationships based on the semantic versioning numbers are as follows:

1.0.0 < 1.1.0 < 1.2.0 < 2.0.0 < 3.0.0 < 3.1.0

1.0.0 < 1.1.0 < 1.1.1_compatible < 1.1.2_non_compatible

1.0.0 < 1.1.0 < 1.2.0 < 1.2.1_non_compatible < 1.2.2_non_compatible

1.0.0 < 1.1.0 < 1.2.0 < 1.3.0 < 1.3.1_non_compatible

1.0.0 < 1.1.0 < 1.2.0 < 1.3.0 < 1.4.0

There is no ordering relationship between "1.1.1_non_compatible" and either "1.2.0" or "1.2.1_non_compatible", except that they share the common ancestor of "1.1.0".

Looking at the version number alone does not indicate ancestry. The module definition in "2.0.0", for example, does not contain all the contents of "1.3.0". Version "2.0.0" is not derived from "1.3.0".

3.4. YANG Semantic Version Update Rules

When a new revision of an artifact is produced, then the following rules define how the YANG semantic version for the new artifact revision is calculated, based on the changes between the two artifact revisions, and the YANG semantic version of the base artifact revision from which the changes are derived.

The following four rules specify the RECOMMENDED, and REQUIRED minimum, update to a YANG semantic version:

1. If an artifact is being updated in a non-backwards-compatible way, then the artifact version "X.Y.Z[_compatible|_non_compatible]" SHOULD be updated to "X+1.0.0" unless that version has already been used for this artifact but with different content, in which case the artifact version "X.Y.Z+1_non_compatible" SHOULD be used instead.

2. If an artifact is being updated in a backwards-compatible way, then the next version number depends on the format of the current version number:

   i "X.Y.Z" - the artifact version SHOULD be updated to "X.Y+1.0", unless that version has already been used for this artifact but with different content, when the artifact version SHOULD be updated to "X.Y.Z+1_compatible" instead.

   ii "X.Y.Z_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_compatible".

   iii "X.Y.Z_non_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_non_compatible".

3. If an artifact is being updated in an editorial way, then the next version number depends on the format of the current version number:

   i "X.Y.Z" - the artifact version SHOULD be updated to "X.Y.Z+1"

   ii "X.Y.Z_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_compatible".

   iii "X.Y.Z_non_compatible" - the artifact version SHOULD be updated to "X.Y.Z+1_non_compatible".

4. YANG artifact semantic version numbers beginning with 0, i.e., "0.X.Y", are regarded as pre-release definitions and need not follow the rules above. Either the MINOR or PATCH version numbers may be updated, regardless of whether the changes are non-backwards-compatible, backwards-compatible, or editorial. See Section 5 for more details on using this notation during module and submodule development.

5. Additional pre-release rules for modules that have had at least one release are specified in Section 5 .

Although artifacts SHOULD be updated according to the rules above, which specify the recommended (and minimum required) update to the version number, the following rules MAY be applied when choosing a new version number:

1. An artifact author MAY update the version number with a more significant update than described by the rules above. For example, an artifact could be given a new MAJOR version number (i.e., X+1.0.0), even though no non-backwards-compatible changes have occurred, or an artifact could be given a new MINOR version number (i.e., X.Y+1.0) even if the changes were only editorial.

2. An artifact author MAY skip version numbers. That is, an artifact's revision history could be 1.0.0, 1.1.0, and 1.3.0 where 1.2.0 is skipped. Note that skipping versions has an impact when importing modules by revision-or-derived. See Section 4 for more details on importing modules with revision-label version gaps.

Although YANG Semver always indicates when a non-backwards-compatible, or backwards-compatible change may have occurred to a YANG artifact, it does not guarantee that such a change has occurred, or that consumers of that YANG artifact will be impacted by the change. Hence, tooling, e.g., [I-D.ietf-netmod-yang-schema-comparison] , also plays an important role for comparing YANG artifacts and calculating the likely impact from changes.

[I-D.ietf-netmod-yang-module-versioning] defines the "rev:non-backwards-compatible" extension statement to indicate where non-backwards-compatible changes have occurred in the module revision history. If a revision entry in a module's revision history includes the "rev:non-backwards-compatible" statement then that MUST be reflected in any YANG semantic version associated with that revision. However, the reverse does not necessarily hold, i.e., if the MAJOR version has been incremented it does not necessarily mean that a "rev:non-backwards-compatible" statement would be present.

3.5. Examples of the YANG Semver Label

3.5.1. Example Module Using YANG Semver

Below is a sample YANG module that uses the YANG Semver revision-label based on the rules defined in this document.

```
module example-versioned-module {
  yang-version 1.1;
  namespace "urn:example:versioned:module";
  prefix "exvermod";
  rev:revision-label-scheme "ysver:yang-semver";

  import ietf-yang-revisions { prefix "rev"; }
  import ietf-yang-semver { prefix "ysver"; }

  description
    "to be completed";

  revision 2017-08-30 {
    description "Backport 'wibble' leaf";
    rev:revision-label 1.2.2_non_compatible;
  }

  revision 2017-07-30 {
    description "Rename 'baz' to 'bar'";
    rev:revision-label 1.2.1_non_compatible;
    rev:non-backwards-compatible;
  }

  revision 2017-04-20 {
    description "Add new functionality, leaf 'baz'";
    rev:revision-label 1.2.0;
  }

  revision 2017-04-03 {
    description "Add new functionality, leaf 'foo'";
    rev:revision-label 1.1.0;
  }

  revision 2017-02-07 {
    description "First release version.";
    rev:revision-label 1.0.0;
  }

  // Note: YANG Semver rules do not apply to 0.X.Y labels.
  // The following pre-release revision statements would not
  // appear in any final published version of a module. They
  // are removed when the final version is published.
  // During the pre-release phase of development, only a
  // single one of these revision statements would appear

  // revision 2017-01-30 {
  //   description "NBC changes to initial revision";
  //   rev:revision-label 0.2.0;
```

```
      //   rev:non-backwards-compatible; // optional
      //                          // (theoretically no
      //                          // 'previous released version')
      // }

      // revision 2017-01-26 {
      //   description "Initial module version";
      //   rev:revision-label 0.1.0;
      // }

      //YANG module definition starts here
    }
```

3.5.2. Example of Package Using YANG Semver

Below is an example YANG package that uses the YANG Semver revision label based on the rules defined in this document.

```
   {
     "ietf-yang-instance-data:instance-data-set": {
       "name": "example-yang-pkg",
       "target-ptr": "TBD",
       "timestamp": "2018-09-06T17:00:00Z",
       "description": "Example IETF package definition",
       "content-data": {
         "ietf-yang-package:yang-package": {
           "name": "example-yang-pkg",
           "version": "1.3.1",
           ...
    }
```

4. Import Module by Semantic Version

[I-D.ietf-netmod-yang-module-versioning] allows for imports to be done based on a module or a derived revision of a module. The rev:revision-or-derived statement can specify either a revision date or a revision label. The YANG Semver revision-label value can be used as the argument to rev:revision-or-derived . When used as such, any module that contains exactly the same YANG semantic version in its revision history may be used to satisfy the import requirement. For example:

```
        import example-module {
          rev:revision-or-derived 3.0.0;
        }
```

Note: the import lookup does not stop when a non-backward-compatible change is encountered. That is, if module B imports a module A at or derived from version 2.0.0, resolving that import will pass through a revision of module A with version "2.1.0_non_compatible" in order to determine if the present instance of module A derives from "2.0.0".

If an import by revision-or-derived cannot locate the specified revision-label in a given module's revision history, that import will fail. This is noted in the case of version gaps. That is, if a module's history includes "1.0.0", "1.1.0", and "1.3.0", an import from revision-or-derived at "1.2.0" will be unable to locate the specified revision entry and thus the import cannot be satisfied.

5. Guidelines for Using Semver During Module Development

This section and the IETF-specific sub-section below provides YANG Semver-specific guidelines to consider when developing new YANG modules. As such this section updates [RFC8407] .

Development of a brand new YANG module or submodule outside of the IETF that uses YANG Semver as its revision-label scheme SHOULD begin with a 0 for the MAJOR version component. This allows the module or submodule to disregard strict SemVer rules with respect to non-backwards-compatible changes during its initial development. However, module or submodule developers MAY choose to use the SemVer pre-release syntax instead with a 1 for the MAJOR version component. For example, an initial module or submodule revision-label might be either 0.0.1 or 1.0.0-alpha.1. If the authors choose to use the 0 MAJOR version component scheme, they MAY switch to the pre-release scheme with a MAJOR version component of 1 when the module or submodule is nearing initial release (e.g., a module's or submodule's revision label may transition from 0.3.0 to 1.0.0-beta.1 to indicate it is more mature and ready for testing).

When using pre-release notation, the format MUST include at least one alphabetic component and MUST end with a '.' or '-' and then one or more digits. These alphanumeric components will be used when deciding pre-release precedence. The following are examples of valid pre-release versions:

1.0.0-alpha.1

1.0.0-alpha.3

2.1.0-beta.42

3.0.0-202007.rc.1

When developing a new revision of an existing module or submodule using the YANG Semver revision-label scheme, the intended target semantic version MUST be used along with pre-release notation. For example, if a released module or submodule which has a current revision-label of 1.0.0 is being modified with the intent to make non-backwards-compatible changes, the first development MAJOR version component must be 2 with some pre-release notation such as -alpha.1, making the version 2.0.0-alpha.1. That said, every publicly available release of a module or submodule MUST have a unique YANG Semver revision-label (where a publicly available release is one that could be implemented by a vendor or consumed by an end user). Therefore, it may be prudent to include the year or year and month development began (e.g., 2.0.0-201907-alpha.1). As a module or submodule undergoes development, it is possible that the original intent changes. For example, a 1.0.0 version of a module or submodule that was destined to become 2.0.0 after a development cycle may have had a scope change such that the final version has no non-backwards-compatible changes and becomes 1.1.0 instead. This change is acceptable to make during the development phase so long as pre-release notation is present in both versions (e.g., 2.0.0-alpha.3 becomes 1.1.0-alpha.4). However, on the next development cycle (after 1.1.0 is released), if again the new target release is 2.0.0, new pre-release components must be used such that every revision-label for a given module or submodule MUST be unique throughout its entire lifecycle (e.g., the first pre-release version might be 2.0.0-202005-alpha.1 if keeping the same year and month notation mentioned above).

5.1. Pre-release Version Precedence

As a module or submodule is developed, the scope of the work may change. That is, while a ratified module or submodule with revision-label 1.0.0 is initially intended to become 2.0.0 in its next ratified version, the scope of work may change such that the final version is 1.1.0. During the development cycle, the pre-release versions could move from 2.0.0-some-pre-release-tag to 1.1.0-some-pre-release-tag. This downwards changing of version numbers makes it difficult to evaluate semantic version rules between pre-release versions. However, taken independently, each pre-release version can be compared to the previously ratified version (e.g., 1.1.0-some-pre-release-tag and 2.0.0-some-pre-release-tag can each be compared to 1.0.0). Module and submodule developers SHOULD maintain only one revision statement in a pre-released module or submodule that reflects the latest revision. IETF authors MAY choose to include an appendix in the associated draft to track overall changes to the module or submodule.

5.2. YANG Semver in IETF Modules

All published IETF modules and submodules MUST use YANG semantic versions for their revision-labels.

Development of a new module or submodule within the IETF SHOULD begin with the 0 MAJOR number scheme as described above. When revising an existing IETF module or submodule, the revision-label MUST use the target (i.e., intended) MAJOR and MINOR version components with a 0 PATCH version component. If the intended ratified release will be non-backward-compatible with the current ratified release, the MINOR version component MUST be 0.

5.2.1. Guidelines for IETF Module Development

All IETF modules and submodules in development MUST use the whole document name as a pre-release version string, including the current document revision. For example, if a module or submodule which is currently released at version 1.0.0 is being revised to include non-backwards-compatible changes in draft-user-netmod-foo, its development revision-labels MUST include 2.0.0-draft-user-netmod-foo followed by the document's revision (e.g., 2.0.0-draft-user-netmod-foo-02). This will ensure each pre-release version is unique across the lifecycle of the module or submodule. Even when using the 0 MAJOR version for initial module or submodule development (where MINOR and PATCH can change), appending the draft name as a pre-release component helps to ensure uniqueness when there are perhaps multiple, parallel efforts creating the same module or submodule.

Some draft revisions may not include an update to the YANG modules or submodules contained in the draft. In that case, those modules or submodules that are not updated do not not require a change to their versions. Updates to the YANG Semver version MUST only be done when the revision of the module changes.

See Appendix A for a detailed example of IETF pre-release versions.

5.2.2. Guidelines for Published IETF Modules

For IETF YANG modules and submodules that have already been published, revision-labels MUST be retroactively applied to all existing revisions when the next new revision is created, starting at version "1.0.0" for the initial published revision, and then incrementing according to the YANG Semver version rules specified in Section 3.4 . For example, if a module or submodule started out in the pre-NMDA ([RFC8342]) world, and then had NMDA support added without removing any legacy "state" branches -- and you are looking to add additional new features -- a sensible choice for the target

YANG Semver would be 1.2.0 (since 1.0.0 would have been the initial, pre-NMDA release, and 1.1.0 would have been the NMDA revision).

6. YANG Module

This YANG module contains the typedef for the YANG semantic version and the identity to signal its use.

```
<CODE BEGINS> file "ietf-yang-semver@2022-09-13.yang"
module ietf-yang-semver {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-semver";
  prefix ysver;
  rev:revision-label-scheme "yang-semver";

  import ietf-yang-revisions {
    prefix rev;
  }

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

     Author:   Joe Clarke
               <mailto:jclarke@cisco.com>
     Author:   Robert Wilton
               <mailto:rwilton@cisco.com>
     Author:   Reshad Rahman
               <mailto:reshad@yahoo.com>
     Author:   Balazs Lengyel
               <mailto:balazs.lengyel@ericsson.com>
     Author:   Jason Sterne
               <mailto:jason.sterne@nokia.com>
     Author:   Benoit Claise
               <mailto:benoit.claise@huawei.com>";
  description
    "This module provides type and grouping definitions for YANG
     packages.

     Copyright (c) 2022 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 Revised 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.: update the date below with the date of RFC publication
    // and remove this note.
    // RFC Ed.: replace XXXX with actual RFC number and remove this
    // note.
    // RFC Ed. update the rev:revision-label to "1.0.0".

    revision 2022-09-13 {
      rev:revision-label "1.0.0-draft-ietf-netmod-yang-semver-08";
      description
        "Initial revision";
      reference
        "RFC XXXX: YANG Semantic Versioning.";
    }

    /*
     * Identities
     */

    identity yang-semver {
      base rev:revision-label-scheme-base;
      description
        "The revision-label scheme corresponds to the YANG Semver
         scheme which is defined by the pattern in the 'version'
         typedef below. The rules governing this revision-label
         scheme are defined in the reference for this identity.";
      reference
        "RFC XXXX: YANG Semantic Versioning.";
    }

    /*
     * Typedefs
     */

    typedef version {
      type rev:revision-label {
        pattern '[0-9]+[.][0-9]+[.][0-9]+(_(non_)?compatible)?'
        + '(-[A-Za-z0-9.-]+[.-][0-9]+)?([+][A-Za-z0-9.-]+)?';
      }
      description
        "Represents a YANG semantic version.  The rules governing the
         use of this revision label scheme are defined in the
         reference for this typedef.";
```

```
      reference
        "RFC XXXX: YANG Semantic Versioning.";
    }
  }
  <CODE ENDS>
```

7. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The design team consists of the following members whom have worked on the YANG versioning project: Balazs Lengyel, Benoit Claise, Bo Wu, Ebben Aries, Jan Lindblad, Jason Sterne, Joe Clarke, Juergen Schoenwaelder, Mahesh Jethanandani, Michael (Wangzitao), Qin Wu, Reshad Rahman, and Rob Wilton.

The initial revision of this document was refactored and built upon [I-D.clacla-netmod-yang-model-update] . We would like the thank Kevin D'Souza for his initial work in this problem space.

Discussions on the use of SemVer for YANG versioning has been held with authors of the OpenConfig YANG models based on their own [openconfigsemver] . We would like thank both Anees Shaikh and Rob Shakir for their input into this problem space.

8. Security Considerations

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

9. IANA Considerations

9.1. YANG Module Registrations

This document requests IANA to register a URI in the "IETF XML Registry" [RFC3688] . Following the format in RFC 3688, the following registration is requested.

URI: urn:ietf:params:xml:ns:yang:ietf-yang-semver

Registrant Contact: The IESG.

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

The following YANG module is requested to be registered in the "IANA Module Names" [RFC6020] . Following the format in RFC 6020, the following registrations are requested:

The ietf-yang-semver module:

Name: ietf-yang-semver

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-semver

Prefix: ysver

Reference: [RFCXXXX]

9.2. Guidance for YANG Semver in IANA maintained YANG modules and submodules

Note for IANA (to be removed by the RFC editor): Please check that the registries and IANA YANG modules and submodules are referenced in the appropriate way.

IANA is responsible for maintaining and versioning some YANG modules and submodules, e.g., iana-if-types.yang [IfTypeYang] and iana-routing-types.yang [RoutingTypesYang] .

In addition to following the rules specified in the IANA Considerations section of [I-D.ietf-netmod-yang-module-versioning] , IANA maintained YANG modules and submodules MUST also include a YANG Semver revision label for all new revisions, as defined in Section 3 .

The YANG Semver version associated with the new revision MUST follow the rules defined in Section 3.4 .

Note: For IANA maintained YANG modules and submodules that have already been published, revision labels MUST be retroactively applied to all existing revisions when the next new revision is created, starting at version "1.0.0" for the initial published revision, and then incrementing according to the YANG Semver rules specified in Section 3.4 .

Most changes to IANA maintained YANG modules and submodules are expected to be backwards-compatible changes and classified as MINOR version changes. The PATCH version may be incremented instead when only editorial changes are made, and the MAJOR version would be incremented if non-backwards-compatible changes are made.

Given that IANA maintained YANG modules are versioned with a linear history, it is anticipated that it should not be necessary to use the "_compatible" or "_non_compatible" modifiers to the "Z_COMPAT" version element.

10. References

10.1. Normative References

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

[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>.

[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>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[I-D.ietf-netmod-yang-module-versioning] Wilton, R., Rahman, R., Lengyel, B., Clarke, J., and J. Sterne, "Updated YANG Module Revision Handling", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-versioning-06, 10 July 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-module-versioning-06.txt>.

10.2. Informative References

[I-D.clacla-netmod-yang-model-update] Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", Work in Progress, Internet-Draft, draft-clacla-netmod-yang-model-update-06, 2 July 2018, <https://www.ietf.org/archive/id/draft-clacla-netmod-yang-model-update-06.txt>.

[I-D.ietf-netmod-yang-packages] Wilton, R., Rahman, R., Clarke, J., Sterne, J., and B. Wu, "YANG Packages", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-packages-03, 4 March 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-packages-03.txt>.

[I-D.ietf-netmod-yang-schema-comparison]
Wilton, R., "YANG Schema Comparison", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-schema-comparison-01, 2 November 2020, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-schema-comparison-01.txt>.

[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>.

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

[SemVer] "Semantic Versioning 2.0.0 (text from June 19, 2020)", <https://github.com/semver/semver/blob/8b2e8eec394948632957639dfa99fc7ec6286911/semver.md>.

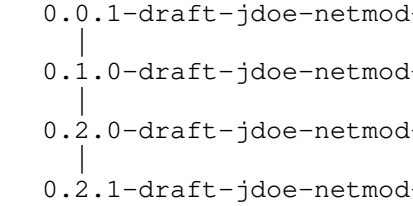
[IfTypeYang]
"iana-if-type YANG Module", <https://www.iana.org/assignments/iana-if-type/iana-if-type.xhtml>.

[RoutingTypesYang]
"iana-routing-types YANG Module", <https://www.iana.org/assignments/iana-routing-types/iana-routing-types.xhtml>.

Appendix A. Example IETF Module Development

Assume a new YANG module is being developed in the netmod working group in the IETF. Initially, this module is being developed in an individual internet draft, draft-jdoe-netmod-example-module. The following represents the initial version tree (i.e., value of revision-label) of the module as it's being initially developed.

Version lineage for initial module development:



At this point, development stabilizes, and the workgroup adopts the draft. Thus now the draft becomes draft-ietf-netmod-example-module. The initial pre-release lineage continues as follows.

Continued version lineage after adoption:

```
1.0.0-draft-ietf-netmod-example-module-00
  |
1.0.0-draft-ietf-netmod-example-module-01
  |
1.0.0-draft-ietf-netmod-example-module-02
```

At this point, the draft is ratified and becomes RFC12345 and the YANG module version becomes 1.0.0.

A time later, the module needs to be revised to add additional capabilities. Development will be done in a backwards-compatible way. Two new individual drafts are proposed to go about adding the capabilities in different ways: draft-jdoe-netmod-exmod-enhancements and draft-asmith-netmod-exmod-changes. These are initially developed in parallel with the following versions.

Parallel development for next module revision (track 1):

```
      1.1.0-draft-jdoe-netmod-exmod-enhancements-00
        |
      1.1.0-draft-jdoe-netmod-exmod-enhancements-01
```

In parallel with (track 2):

```
      1.1.0-draft-asmith-netmod-exmod-changes-00
        |
      1.1.0-draft-asmith-netmod-exmod-changes-01
```

At this point, the WG decides to merge some aspects of both and adopt the work in asmith's draft as draft-ietf-netmod-exmod-changes. A single version lineage continues.

```
      1.1.0-draft-ietf-netmod-exmod-changes-00
        |
      1.1.0-draft-ietf-netmod-exmod-changes-01
        |
      1.1.0-draft-ietf-netmod-exmod-changes-02
        |
      1.1.0-draft-ietf-netmod-exmod-changes-03
```

The draft is ratified, and the new module version becomes 1.1.0.

Authors' Addresses

Joe Clarke (editor)
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

Robert Wilton (editor)
Cisco Systems, Inc.
Email: rwilton@cisco.com

Reshad Rahman
Email: reshad@yahoo.com

Balazs Lengyel
Ericsson
1117 Budapest
Magyar Tudosok Korutja
Hungary
Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Jason Sterne
Nokia
Email: jason.sterne@nokia.com

Benoit Claise
Huawei
Email: benoit.claise@huawei.com

NETMOD
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2023

Q. Ma
Q. Wu
Huawei
B. Lengyel
Ericsson
H. Li
HPE

20 October 2022

YANG Extension and Metadata Annotation for Immutable Flag
draft-ma-netmod-immutable-flag-04

Abstract

This document defines a YANG extension named "immutable" to indicate that specific "config true" data nodes are not allowed to be created/deleted/updated. To indicate that specific entries of a list/leaf-list node or instances inside list entries cannot be updated/deleted after initialization, a metadata annotation with the same name is also defined. Any data node or instance marked as immutable is read-only to the clients of YANG-driven management protocols, such as NETCONF, RESTCONF and other management operations (e.g., SNMP and CLI requests).

This document aims to provide more visibility into immutability characteristic of particular schema or instance nodes by defining a standard mechanism to allow the server to document the existing immutable configuration data, while this doesn't mean attaching such restrictions is encouraged.

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 23 April 2023.

Table of Contents

1. Introduction

YANG [RFC7950] is a data modeling language used to model both state and configuration data, based on the "config" statement. However there exists data that cannot be modified by the client, but still needs to be declared as "config true" to:

* allow configuration of data nodes under immutable lists or containers;

* ensure the existence of specific list entries that are provided and needed by the system, while additional list entries can be created, modified or deleted;

* place "when", "must" and "leafref" constraints between configuration and immutable schema nodes.

For example, the interface type value created by the system due to the hardware currently present in the device cannot be modified by clients, while configurations such as MTU created by the system are free to be modified by the client. Further examples and use-cases are described in Appendix A.

Allowing some configuration to be modifiable while other parts are not is inconsistent and introduces ambiguity to clients.

To address this issue, this document defines a YANG extension named "immutable" to indicate that specific "config true" data nodes are not allowed to be created/deleted/updated. To indicate that specific entries of a list/leaf-list node or instances inside list entries cannot be updated/deleted after initialization, a metadata annotation [RFC7952] with the same name is also defined. Any data node or instance marked as immutable is read-only to the clients of YANG-driven management protocols, such as NETCONF, RESTCONF and other management operations (e.g., SNMP and CLI requests). Marking instance data nodes as immutable (as opposed to marking schema-nodes) is useful when only some instances of a list or leaf-list shall be marked as read-only.

It is already the case that a server can reject any configuration for any reason, e.g., when a client tries to modify an immutable configuration data. This document aims to provide more visibility into immutability characteristic of particular schema or instance nodes by defining a standard mechanism to allow the server to document the existing immutable configuration data, while this doesn’t mean attaching such restrictions is encouraged.

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.

The following terms are defined in [RFC6241] and [RFC8341] and are not redefined here:

* configuration data

* access operation

* write access

The following terms are defined in this document:

immutable: A schema or instance node property indicating that the configuration data is not allowed to be created/deleted/updated.

2. Overview

The "immutable" concept defined in this document only indicates write access restrictions to read-write datastores. A particular data node or instance MUST have the same immutability in all read-write datastores. The immutable annotation information should also be visible even in read-only datastores (e.g., <system>, <intended>, <operational>), however this only serves as information about the data node itself, but has no effect on the handling of the read-only datastore. The immutability property of a particular data node or instance MUST be protocol-independent and user-independent.

If a particular leaf-list node is marked as "immutable" without exceptions for "delete" in the schema, the server SHOULD NOT annotate its instances, as that provides no additional information. If a particular container/list/leaf/anydata/anyxml node is marked as "immutable" without exceptions for "delete" or "update" in the schema, the server SHOULD NOT annotate its instances, as that provides no additional information.

Already today the server rejects any attempt to the "create", "delete" or "update" access operations on an immutable configuration data. This document allows the existing immutable data node or instance to be marked by YANG extension or metadata annotation. Requests to create/update/delete an immutable configuration data always return an error (except the exceptions argument in YANG extension). The error reporting is performed immediately at an <edit-config> operation time, regardless what the target configuration datastore is. For an example of an "invalid-value" error response, see Appendix A.1.2.

However the following operations SHOULD be allowed:

* Use a create, update, delete/remove operation on an immutable node if the effective change is null. E.g., if a leaf has a current value of "5" it should be allowed to replace it with a value of "5";

* Create an immutable node with a same value initially set by the system if it doesn't exist in the datastore. E.g., it should be allowed to explicitly configure a system-defined interface name and type in <running> as the same values in <system>;

* Delete an immutable node in <running> which is instantiated in <system> and copied into <running>, unless the resource is no longer available (e.g., the interface removed physically), there is no way to actually delete system configuration from a server [I-D.ma-netmod-with-system], even if the node in the schema tree is declared as "immutable" without the exception for "delete".

Note that even if a particular data node is immutable without the exception for "delete", it still can be deleted with its parent node, e.g., /if:interfaces/if:interface/if:type leaf is immutable, but the deletion to the /if:interfaces/if:interface list entry is allowed; if a particular data node is immutable without the exception for "create", it means the client can never create the instance of it, regardless the handling of its parent node.

When a specific data node or instance is marked as "immutable", NACM cannot override this to allow create/delete/update access. Servers will ignore such NACM rule. For example, if a particular data node is marked as "im:immutable" without the "exceptions" argument for update, the server will ignore any user-defined NACM rule to allow update access operation to that specific data node.

Write access restriction due to general YANG rules has no need to be marked as immutable. For example, key leaf which is given a value when a list entry is created cannot be modified and deleted unless the list entry is deleted. A mandatory leaf MUST exist and cannot be deleted if the ancestor node exists in the data tree. Decorating the key leaf and mandatory leaf as immutable provides no additional information in these cases.

3. "Immutable" YANG Extension

The "immutable" YANG extension can be a substatement to a "config true" leaf, leaf-list, container, list, anydata or anyxml statement. It indicates that data nodes based on the parent statement are not allowed to be added, removed or updated except according to the exceptions argument. Any such write attempt will be rejected by the server.

The "immutable" YANG extension defines an argument statement named "exceptions" which gives a list of operations that users are permitted to invoke for the specified node.

The following values are supported for the "exceptions" argument:

* Create: allow users to create instances of the data node;

* Update: allow users to modify instances of the data node;

* Delete: allow users to delete instances of the data node.

If more than one value are intended, a space-separated string for the "exceptions" argument is used. For example, if the instance of a particular data node can always be created and modified, it cannot be deleted, the following "immutable" YANG extension with "create" and "update" exceptions could be defined in a substatement to that data node:

```
im:immutable "create update";
```

Providing an empty string for the "exceptions" argument is equivalent to a single extension without an argument followed. Providing all 3 values has the same effect as not using this exceptions at all, but can be used anyway.

Note that leaf-list instances can be created and deleted, but not modified. Any exception for "update" operation to leaf-list data nodes SHOULD be ignored.

4. "Immutable" Metadata Annotation

The "immutable" flag is used to indicate the immutability of a particular instantiated data node. It only applies to the list/leaf-list entries or instances inside particular list entries. The values are boolean types indicating whether the data node instance is immutable or not.

Note that "immutable" metadata annotation is used to annotate instances of a list/leaf-list rather than schema nodes. For instance, a list node may exist in multiple instances in the data tree, "immutable" can annotate some of the instances as read-only, while others are not.

Any list/leaf-list instance annotated with immutable="true" is read-only to clients, which means that once an instance is created, the client cannot update/delete it. If a list entry is annotated with immutable="true", any contained descendant instances of any type

(including leafs, lists, containers, etc.) inside the specific instance is not allowed to be created, updated and deleted without the need to annotate descendant nodes instances explicitly.

When the client retrieves a particular datastore, immutable data node instances MUST be annotated with immutable="true" by the server. If the "immutable" metadata annotation for a list/leaf-list entry is not specified, the default "immutable" value is false.

5. Inheritance of Immutability

Comment: This section tries to answer the questions : Is immutable inherited down the containment hierarchy? If it is, should we allow overriding the immutability of a particular contained element (i.e., to declare a contained data node as immutable=false inside an immutable container/list) ?

Unless otherwise specified, the immutability for data nodes is inherited from their parent nodes. The immutability in the hierarchy is inherited downwards towards the leaf/leaf-list nodes. Specifically, if a node has child elements, non-modification to that node means any child elements is not allowed to be create, update and delete. E.g., if a particular instance of a list node is not allowed to be updated, any descendant node instance is not allowed to be create, update and delete inside that list instance. In this case, there is no need to mark the descendant nodes as immutable.

For the following YANG example:

```
list role {
  key name;
  leaf name {
    type string;
  }
  leaf-list granted-operation {
    type enumeration {
      enum read;
      enum write;
      enum execute;
      enum debug;
    }
  }
}
```

A system-defined corresponding XML instance with immutable annotation example:

```
    <role im:immutable="rue">
      <name>owner</name>
      <granted-operation>read</granted-operation>
      <granted-operation>write</granted-operation>
      <granted-operation>execute</granted-operation>
      <granted-operation>debug</granted-operation>
    </role>
```

The role instance named "owner" is annotated with immutable="true", which means that this instance is not allowed to be updated and deleted, all of the child nodes are not allowed to be created, updated and deleted inside this instance. For example, if a client tries to delete an existing granted-operation "debug" inside the "owner" role in an <edit-config> operation, an "operation-not-supported" error is returned.

However, sometimes there is a desire to override the immutability of a particular contained node. For example, given the following list definition:

```
  list application {
    im:immutable "create delete";
    key name;
    leaf name {
      type string;
    }
    leaf protocol {
      type enumeration {
        enum tcp;
        enum udp;
      }
    }
    leaf port-number {
      im:immutable "update";
      type string;
    }
  }
```

Any list entries of the list node "application" is allowed to be created and deleted, but not modification. However, the contained leaf node "port-number" has the immutability with exception for "update" operation, this means that modification to the value of leaf node "port-number" inside "application" list instance is allowed.

6. YANG Module

```
   <CODE BEGINS>
    file="ietf-immutable@2022-08-11.yang"
   // RFC Ed.: replace XXXX with RFC number and remove this note
     module ietf-immutable {
       yang-version 1.1;
       namespace "urn:ietf:params:xml:ns:yang:ietf-immutable";
       prefix im;

       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: Qiufang Ma
                  <mailto:maqiufang1@huawei.com>

          Author: Qin Wu
                  <mailto:bill.wu@huawei.com>

          Author: Balazs Lengyel
                  <mailto:balazs.lengyel@ericsson.com>

          Author: Hongwei Li
                  <mailto:flycoolman@gmail.com>";

       description
         "This module defines a metadata annotation named 'immutable'
          to indicate the immutability of a particular instantiated
          data node. Any instantiated data node marked with
          immutable='true' by the server is read-only to the clients
          of YANG-driven management protocols, such as NETCONF,
          RESTCONF as well as SNMP and CLI requests.

          The module defines the immutable extension that indicates
          that data nodes based on data-definition statement cannot
          be added removed or updated except according to the
          exceptions argument.

          Copyright (c) 2022 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 Revised
          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 HHHH
          (https://www.rfc-editor.org/info/rfcHHHH); 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 2022-08-11 {
        description
          "Initial revision.";
        reference
          "RFC XXXX: YANG Extension and Metadata Annotation for Immutable Flag";
      }

      extension immutable {
        argument exceptions;
        description
          "The 'immutable' extension as a substatement to a data
           definition statement indicates that data nodes based on
           the parent statement MUST NOT be added, removed or
           updated by management protocols, such as NETCONF,
           RESTCONF or other management operations (e.g., SNMP
           and CLI requests) except when indicated by the
           exceptions argument.

           Immutable data MAY be marked as config true to allow
           'leafref', 'when' or 'must' constraints to be based
           on it.

           The statement MUST only be a substatement of the leaf,
           leaf-list, container, list, anydata, anyxml statements.
           Zero or one immutable statement per parent statement
           is allowed.
           No substatements are allowed.

           The argument is a list of operations that are
           permitted to be used for the specified node, while
```

```
            other operations are forbidden by the immutable extension.
            - create: allows users to create instances of the data node
            - update: allows users to modify instances of the data node
            - delete: allows users to delete instances of the data node

            To disallow all user write access, omit the argument;

            To allow only create and delete user access, provide
            the string 'create delete' for the 'exceptions' parameter.

            Providing all 3 parameters has the same effect as not
            using this extension at all, but can be used anyway.

            Equivalent YANG definition for this extension:

            leaf immutable {
              type bits {
                bit create;
                bit update;
                bit delete;
              }
              default '';
            }


            Adding immutable or removing values from the
            exceptions argument of an existing immutable statement
            are non-backwards compatible changes.
            Other changes to immutable are backwards compatible.";
        }

        md:annotation immutable {
          type boolean;
          description
            "The 'immutable' annotation indicates the immutability of an
             instantiated data node. Any data node instance marked as
             'immutable=true' is read-only to clients and cannot be
             updated through NETCONF, RESTCONF or CLI. It applies to the
             list and leaf-list entries. The default is 'immutable=false'
             if not specified for an instance.";
        }
      }
   <CODE ENDS>
```

7. IANA Considerations

7.1. The "IETF XML" Registry

This document registers one XML namespace URN in the 'IETF XML registry', following the format defined in [RFC3688].

URI: urn:ietf:params:xml:ns:yang:ietf-immutable
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

7.2. The "YANG Module Names" Registry

This document registers one module name in the 'YANG Module Names' registry, defined in [RFC6020].

name: ietf-immutable
prefix: im
namespace: urn:ietf:params:xml:ns:yang:ietf-immutable
RFC: XXXX // RFC Ed.: replace XXXX and remove this comment

8. Security Considerations

The YANG module specified in this document defines a metadata annotation for data nodes that is designed to be accessed 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 [RFC8446].

Since immutable information is tied to applied configuration values, it is only accessible to clients that have the permissions to read the applied configuration values.

The security considerations for the Defining and Using Metadata with YANG (see Section 9 of [RFC7952]) apply to the metadata annotation defined in this document.

Acknowledgements

Thanks to Kent Watsen, Andy Bierman, Robert Wilton, Jan Lindblad, Reshad Rahman, Anthony Somerset, Lou Berger, Joe Clarke for reviewing, and providing important input to, this document.

References

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>.

[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>.

[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>.

[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>.

Informative References

[I-D.ma-netmod-with-system]
Ma, Q., Wu, Q., and C. Feng, "System-defined Configuration", Work in Progress, Internet-Draft, draft-ma-netmod-with-system-05, 29 September 2022, <https://www.ietf.org/archive/id/draft-ma-netmod-with-system-05.txt>.

[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>.

Appendix A. Usage Examples

A.1. Interface Example

This section shows how to use im:immutable YANG extension to mark some data node as immutable.

When an interface is physically present, the system will create an interface entry automatically with valid name and type values in <system> (see [I-D.ma-netmod-with-system]). The system-generated data is dependent on and must represent the HW present, and as a consequence must not be changed by the client. The data is modelled as "config true" and should be marked as immutable.

Seemingly an alternative would be to model the list and these leaves as "config false", but that does not work because:

* The list cannot be marked as "config false", because it needs to contain configurable child nodes, e.g., ip-address or enabled;

* The key leaf (name) cannot be marked as "config false" as the list itself is config true;

* The type cannot be marked "config false", because we MAY need to reference the type to make different configuration nodes conditionally available.

The immutability of the data is the same for all interface instances, thus following fragment of a fictional interface module including an "immutable" YANG extension can be used:

```
        container interfaces {
          list interface {
            key "name";
            leaf name {
              type string;
            }
            leaf type {
              im:immutable "create";
              type identityref {
                base ianaift:iana-interface-type;
              }
              mandatory true;
            }
            leaf mtu {
              type uint16;
            }
            leaf-list ip-address {
              type inet:ip-address;
            }
          }
        }
```

Note that the "name" leaf is defined as a list key which can never been modified for a particular list entry, there is no need to mark "name" as immutable.

A.1.1. Creating an Interface with a "type" Value

As defined in the YANG model, there is an exception for "create" operation. Assume the interface hardware is not present physically at this point, the client is allowed to create an interface named "eth0" with a type value in <running>:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interface xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0"
            xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
            xc:operation="create">
        <name>eth0</name>
        <type>ianaift:ethernetCsmacd</type>
      </interface>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

The interface data does not appear in <operational> since the physical interface doesn't exist. When the interface is inserted, the system will detect it and create the associated configuration in <system>. The system tries to merge the interface configuration in the <running> datastore with the same name as the inserted interface configuration in <system>. If no such interface configuration named "eth0" is found in <system> or the type set by the client doesn't match the real interface type generated by the system, only the system-defined interface configuration is applied and present in <operational>.

A.1.2. Updating the Value of an Interface Type

Assume the system applied the interface configuration named "eth0" successfully. If a client tries to change the type of an interface to a value that doesn't match the real type of the interface used by the system, the request will be rejected by the server:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <interface xc:operation="merge"
            xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
        <name>eth0</name>
        <type>ianaift:tunnel</type>
      </interface>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path xmlns:t="http://example.com/schema/1.2/config">
      /interfaces/interface[name="eth0"]/type
    </error-path>
    <error-message xml:lang="en">
      Invalid type for interface eth0
    </error-message>
  </rpc-error>
</rpc-reply>
```

A.2. Immutable System Capabilities Modelled as "config true"

System capabilities might be represented as system-defined data nodes in the model. Configurable data nodes might need constraints specified as "when", "must" or "path" statements to ensure that configuration is set according to the system's capabilities. E.g.,

* A timer can support the values 1,5,8 seconds. This is defined in the leaf-list 'supported-timer-values'.

* When the configurable 'interface-timer' leaf is set, it should be ensured that one of the supported values is used. The natural solution would be to make the 'interface-timer' a leaf-ref pointing at the 'supported-timer-values'.

However, this is not possible as 'supported-timer-values' must be read-only thus config=false while 'interface-timer' must be writable thus config=true. According to the rules of YANG it is not allowed to put a constraint between config true and false schema nodes.

The solution is that the supported-timer-values data node in the YANG Model shall be defined as "config true" and shall also be marked with the "immutable" extension. After this the 'interface-timer' shall be defined as a leaf-ref pointing at the 'supported-timer-values'.

A.3. Immutable System-defined List Entries

There are some system-defined entries for a "config true" list which are present in <system> (see [I-D.ma-netmod-with-system]) and cannot be updated by the client, such system-defined instances should be defined immutable. The client is free to define, update and delete their own list entries in <running>. Thus the list data node in the YANG model cannot be marked as "immutable" extension as a whole. But some of the system-defined list entries need to be protected if they are copied from the <system> datastore to <running>.

An immutable metadata annotation can be useful in this case. When the client retrieves those system-defined entries towards <system> (or <running> if they are copied into <running>), an immutable="true" annotation is returned; so that the client can understand that the predefined list entries shall not be updated but they can configure their list entries without any restriction.

Appendix B. Changes between revisions

Note to RFC Editor (To be removed by RFC Editor)

v03 - v04

* Clarify how immutable flag interacts with NACM mechanism.

v02 - v03

* rephrase and avoid using "server MUST reject" statement, and try to clarify that this documents aims to provide visibility into existing immutable behavior;

* Add a new section to discuss the inheritance of immutability;

* Clarify that deletion to an immutable node in <running> which is instantiated in <system> and copied into <running> should always be allowed;

* Clarify that write access restriction due to general YANG rules has no need to be marked as immutable.

* Add an new section named "Acknowledgements";

* editoral changes.

v01 - v02

* clarify the relation between the creation/deletion of the immutable data node with its parent data node;

* Add a "TODO" comment about the inheritance of the immutable property;

* Define that the server should reject write attempt to the immutable data node at an <edit-config> operation time, rather than waiting until a <commit> or <validate> operation takes place;

v00 - v01

* Added immutable extension

* Added new use-cases for immutable extension and annotation

* Added requirement that an update that means no effective change should always be allowed

* Added clarification that immutable is only applied to read-write datastore

* Narrowed the applied scope of metadata annotation to list/leaf-list instances

Appendix C. Open Issues tracking

* Can we do better about the "immutable" terminology?

* Is the preferred solution best?

Authors' Addresses

Qiufang Ma
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China

Email: maqiufang1@huawei.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com

Balazs Lengyel
Ericsson
Email: balazs.lengyel@ericsson.com

Hongwei Li
HPE
Email: flycoolman@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 26 April 2023

Q. Ma
Q. Wu
Huawei
M. Boucadair
Orange
D. King
Old Dog Consulting
23 October 2022

A Policy-based Network Access Control
draft-ma-opsawg-ucl-acl-00

Abstract

This document defines two YANG modules for policy-based network access control, which provide consistent and efficient enforcement of network access control policies based on user-group identity. In addition, this document defines a mechanism to ease the maintenance of the mapping between a user-group ID and a set of IP/MAC addresses to enforce policy-based network access control. Finally, the document defines a RADIUS attribute that is used to communicate the user group identifier as part of identification and authorization information.

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 26 April 2023.

Copyright Notice

Table of Contents

1. Introduction

With the increased adoption of remote access technologies (e.g., Virtual Private Networks (VPNs)), Small- and Medium-size Businesses (SMBs) are increasingly adopting cloud-based security services to replace or complement on-premises security tools. Such tools are meant to facilitate access to Enterprise resources for authorized users from anywhere. However, from a technical standpoint, enabling large-scale employee mobility across many access locations induces a set of challenges compared to conventional network access management approaches, e.g.:

* Endpoints do not have a stable IP address. For example, Wireless LAN (WLAN) and VPN clients, as well as back-end Virtual Machine (VM)-based servers, can move; their IP addresses could change as a result. This means that relying on IP/transport fields (e.g., the 5-tuple) is inadequate to ensure consistent and efficient security policy enforcement. IP address-based policies may not be flexible enough to accommodate endpoints with volatile IP addresses.

* With the massive adoption of teleworking, there is now a need to apply different security policies to the same set of users under different circumstances (e.g., prevent relaying attacks against a local attachment point to the Enterprise network). For example, network access might be granted based upon criteria such as users' location, source network reputation, users' role, time-of-day, type of network device used (e.g., corporate issued device versus personal device), device's security posture, etc. This means the network needs to recognize the users' identity and their current context, and map the users to their correct access entitlement to the network.

This document defines two YANG modules for policy-based Network Access Control [NIST-ABAC]. These modules are meant to ensure consistent enforcement of access control policies based on user-group identity. In addition, this document defines a mechanism to establish mapping between the user-group ID and IP/MAC addresses to execute the policy-based access control.

Also, the document defines a RADIUS attribute that used to communicate the user group identifier as part of identification and authorization information (Section 6).

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.

The meanings of the symbols in tree diagrams are defined in [RFC8340].

The document uses the terms defined in [RFC8519].

In the current version of the draft, the term "user" refers also to the host that actually connect to a network. Also, The term "user" refers to any user of the network. As such, servers, terminals, and

other devices are also classified and assigned to their respective user-groups. Future versions of the document will call out specifically whether a user or a user's host are concerned.

3. Sample Usage

The network needs to recognize the users' identities regardless of the change of the IP addresses of the device they use to connected to the network. Then, the network maps the users to their access authorization rights. As discussed in the introduction, because there is a large number of users and the IP addresses of the same user are in different network segments, deploying a network access control policy for each IP address or network segment is heavy workload. An alternative approach is to configure user groups to classify users (and their devices) and associate ACLs with user groups so that users in each group can share a group of ACL rules. This approach greatly reduces the workload of the administrator and optimizes the ACL resources on the device that behaves as a PEP (Policy Enforcement Point) [RFC3198]. The Network ACLs (NACLs) can be provisioned on devices using specific mechanisms, such as [RFC8519] or [I-D.dbb-netmod-acl].

Network access control policies may need to vary over time. For example, companies may restrict/grant employees access to specific resources (internal and/or external resources) during work hours, while another policy is adopted during off-hours and weekends. A network administrator may also require to enforce traffic shaping (Section 2.3.3.3 of [RFC2475]) and policing (Section 2.3.3.4 of [RFC2475]) during peak hours in order not to affect other data services.

4. Policy-based Network Access Control: An Overview

To provide real-time and consistent enforcement of access control policies, the following functional entities and interfaces are involved:

* A Service Orchestrator which coordinates the overall service, including security policies.

* A Policy Decision Point (PDP) [RFC8519] maintains access permissions associated with different user groups, inter groups, and aggregates information about device types, access locations, and other attribute information. One or multiple PDPs can be deployed in a network. The inter-user-group access permissions describe user group to group communication policies.

* The Security Controller is responsible for implementing the YANG module defined in Section 5.1 and maintains the user-to-"user-group" mapping with attributes information. If necessary, the Security Controller retrieves the access permissions from the PDP and pushes the required user-group access control policies to relevant PEPs that need them.

  The Security Controller exposes a RESTCONF [RFC8040] or NETCONF [RFC6241] interface to the PDP.

* A User Authentication Device (UAD) entity which handles authentication requests. When access is granted, the UAD provides the group identifier (group ID) to which the user belongs when the user first logs onto the network. The UAD interacts with a AAA server to complete user authentication using RADIUS [RFC2865]. A new attribute is defined in Section 6.

* The PEP is the central entity which is responsible for implementing the YANG module defined in Section 5.2 and maintaining Access Control Lists, and enforcing appropriate policies. A PEP maps incoming packets to their associated user-group IDs, and acts on the user-group IDs. The policies are expressed as user-group (not IP or MAC address) IDs so as to decouple the user identity from the network addresses of the user's device. If the PEP also co-locates with the user authentication device, it maintains the mapping between the user-group IDs and the IP or MAC address.

  Multiple PEPs can be involved in a network.

  A PEP exposes a NETCONF interface to the Security Controller.

  A PEP may be collocated with a UAD.

Figure 1 provides the overall architecture and procedure for policy-based access control management.

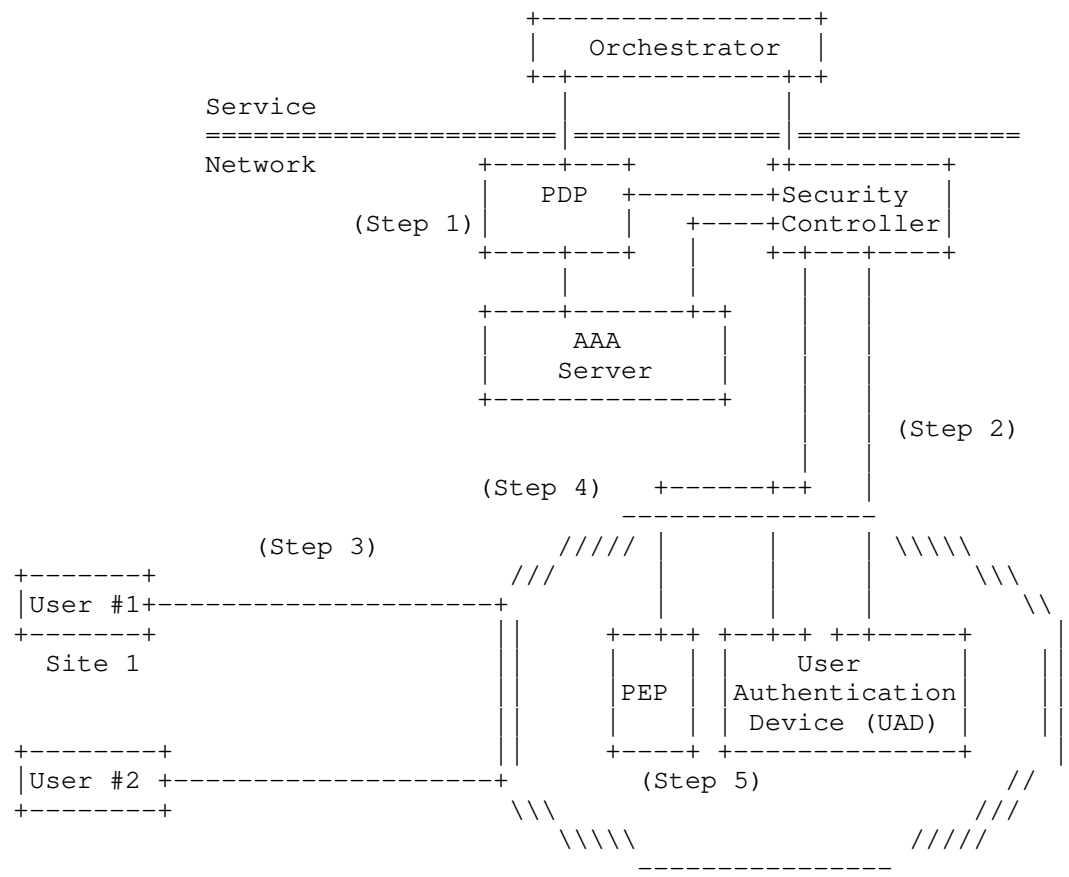


Figure 1: An Architecture for Group-based Policy Management

In reference to Figure 1, the following typical flow is experienced:

Step 1: Administrators (or the Orchestrator) configure the users, user-groups, and related attribute information on the Security Controller using the YANG module defined in Section 5.1. The inter-user-group and group-to-group access permissions are also managed by administrators and maintained by the PDP.

Step 2: The user-group-based access control policies are maintained on relevant PEPs under the controller's management. This may require obtaining access control permissions and attribute information from the PDP and an AAA server. This is implemented via the Security Controller.

Step 3: When a user first logs onto the network, the user is

required to be authenticated (e.g., using user name and password) at the UAD.

Step 4: The authentication request is then relayed to the AAA server (see Section 6). If the authentication request succeeds, the user is placed in a user-group, as determined by the PDP and the user-group ID is returned to the user authentication device as the authentication result. The UAD also records the mapping between the user-group IDs and the IP or MAC address and reports to the controller. If the authentication fails, then the user is not assigned a user-group, which also means that the user has no or limited access permissions for the network. ACLs are enforced so that flows from that IP address are discarded by the network.

Step 5: The user's subsequent traffic is allowed or permitted based on the user-group based access control policies maintained by the PEP, during which process PEP matches common header information, such as n-tuple and then maps it to the user-group ID . If the PEP is also the UAD, it already maintains the mapping information. Otherwise, it requests the mapping information from the controller.

4.1. User Groups

The user-group ID is an identifier that represents the collective identity of a group of users. It is determined by a set of pre-defined policy criteria (e.g., source IP address, geolocation data, time of day, or device certificate). Users may be moved to different user-groups if their composite attributes, environment, and/or local enterprise policy change.

A user is authenticated, and classified at the network ingress, and assigned to a user-group. A user's group membership may change as aspects of the user change. For example, if the user-group membership is determined solely by the source IP address, then a given user's user-group ID will change when the user moves to a new IP address that falls outside of the range of addresses of the previous user-group.

This document does not make an assumption how groups are defined. Such considerations are deployment specific and are out of scope. However, and for illustration purposes, Table 1 shows an example of how user-group definitions may be characterized. User-groups may share several common criteria. That is, user-group criteria are not mutually exclusive. For example, the policy criteria of user-groups R&D Regular and R&D BYOD may share the same set of users that belong to the R&D organization, and differ only in the type of clients (firm-issued clients vs. users' personal clients). Likewise, the same user may be assigned to different user-groups depending on the time of day or the type of day (e.g., weekdays versus weekends), etc.

Group Name	Group ID	Group Definition
R&D	10	R&D employees
R&D BYOD	11	Personal devices of R&D employees
Sales	20	Sales employees
VIP	30	VIP employees
Workflow	40	IP addresses of Workflow resource servers
R&D Resource	50	IP addresses of R&D resource servers
Sales Resource	54	IP addresses of Sales resource servers

Figure 2: Table 1: User-Group Example

5. YANG Modules

5.1. The UCL Group YANG Module

5.1.1. Module Overview

Figure 3 provide an overview of the tree structure of the "ietf-ucl-group" module.

```
module: ietf-ucl-group
  +--rw ucl-groups
     +--rw user-group* [group-id]
        +--rw group-id    uint32
        +--rw role?       identityref
        +--rw user* [user-name]
           +--rw user-name                    string
           +--rw address-grouping-mapping
           |  +--rw address* [address-id]
           |     +--rw address-id      uint32
           |     +--rw ipv4-address?   inet:ipv4-prefix
           |     +--rw ipv6-address?   inet:ipv6-prefix
           |     +--rw mac-address?    yang:mac-address
           +--rw access-locations
           |  +--rw location* [location-id]
           |     +--rw location-id    string
           |     +--rw address?       string
           |     +--rw postal-code?   string
           +--rw accessed-devices?           identityref
           +--rw start-time?                 yang:date-and-time
           +--rw end-time?                   yang:date-and-time
```

Figure 3: UCL Tree Diagram

This module is defined as a standalone module and used to establish on the Security Controller the mapping between group-id and associated attributes such as role, location, IP address, MAC address, accessed resources, access period. Attributes are assigned to specific users, and then determine access based on those attributes. These attributes could include a user's position or role, but may also include their location, the time of day, and other factors.

5.1.2. The YANG Module

This module imports [RFC6991].

```
<CODE BEGINS>
 file="ietf-ucl-group@2022-10-14.yang"
module ietf-ucl-group {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ucl-group";
  prefix uclg;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types, Section 3";
```

```
}
import ietf-inet-types {
  prefix inet;
  reference
    "RFC 6991: Common YANG Data Types, Section 4";
}

organization
  "IETF OPSAWG Working Group";
contact
  "WG Web: <https://datatracker.ietf.org/wg/opsawg/>
   WG List: <mailto:opsawg@ietf.org>";
description
  "This YANG module defines XXX.

   Copyright (c) 2022 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 Revised
   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 2022-10-14 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A Policy-based Network Access Control";
}

identity device-type {
  description
    "Base identity for device type.";
}

identity role-type {
  description
    "Identity for role group type.";
}

identity smartphone {
  base device-type;
```

```
    description
      "Identity for the smartphone terminal device.";
  }

  identity tablet {
    base device-type;
    description
      "Identity for the tablet accessed device.";
  }

  identity laptop {
    base device-type;
    description
      "Identity for the laptop accessed device.";
  }

  identity pc {
    base device-type;
    description
      "Identity for the PC accessed device.";
  }

  identity finance {
    base role-type;
    description
      "Identity for the finance role.";
  }

  identity sales {
    base role-type;
    description
      "Identity for the sales role.";
  }

  identity research {
    base role-type;
    description
      "Identity for the research role.";
  }

  identity developer {
    base role-type;
    description
      "Identity for the developer role.";
  }

  identity vip {
    base role-type;
```

```
      description
        "Identity for the VIP role.";
    }

    identity visitor {
      base role-type;
      description
        "Identity for the guest role.";
    }

    container ucl-groups {
      description
        "Defines the UCL groups";
      list user-group {
        key "group-id";
        description
          "The user group with which the traffic flow is
           associated can be identified by a user-group id.";
        leaf group-id {
          type uint32;
          description
            "The ID of the group which is used to
             identified a user group. This identifier is
             unique within the scope of a network.";
        }
        leaf role {
          type identityref {
            base role-type;
          }
          description
            "The common role of this user-group.";
        }
        list user {
          key "user-name";
          description
            "List of users indexed by their user-name.";
          leaf user-name {
            type string {
              length "1..max";
            }
            description
              "A special name given to a user to uniquely identify them.";
          }
          container address-grouping-mapping {
            description
              "Defines lists of IP and MAC addresses.";
            list address {
              key "address-id";
```

```
            description
              "The possible accessed address of the user, identified
               by the address-id.";
            leaf address-id {
              type uint32;
              description
                "A unique address-id that identifies a user's accessed
                 address.";
            }
            leaf ipv4-address {
              type inet:ipv4-prefix;
              description
                "The IPv4 address prefix of the user's accessed IP.";
            }
            leaf ipv6-address {
              type inet:ipv6-prefix;
              description
                "The IPv6 address prefix of the user's accessed IP.";
            }
            leaf mac-address {
              type yang:mac-address;
              description
                "The mac address of the user's accessed device.";
            }
          }
        }
        container access-locations {
          description
            "Defines lists of locations.";
          list location {
            key "location-id";
            description
             "List of locations.";
            leaf location-id {
              type string {
                length "1..max";
              }
              description
                "Location id information.";
            }
            leaf address {
              type string;
              description
                "User detailed address information.";
            }
            leaf postal-code {
              type string;
              description
```

```
                    "Postal code information of the user's
                     accessed location.";
                }
              }
            }
            leaf accessed-devices {
              type identityref {
                base device-type;
              }
              description
                "The user's accessed device type.";
            }
            leaf start-time {
              type yang:date-and-time;
              description
                "The start time that the user belongs to
                 this group ID.";
            }
            leaf end-time {
              type yang:date-and-time;
              description
                "The end time that the user belongs to
                 this group ID.";
            }
          }
        }
      }
    }
    <CODE ENDS>
```

5.2. The UCL Extension to the ACL Model

5.2.1. Module Overview

Figure 4 provides the tree strcuture of the "ietf-ucl-acl" module.

```
module: ietf-ucl-acl
  augment /acl:acls/acl:acl/acl:aces/acl:ace/acl:matches:
    +--rw (user-control-groups)? {match-on-user-group}?
       +--:(source-match)
       |  +--rw source-match
       |     +--rw (match)?
       |        +--:(user-group)
       |        |  +--rw user-group-id?   uint32
       |        +--:(IP-address)
       |           +--rw ipv4-network?    inet:ipv4-prefix
       |           +--rw ipv6-network?    inet:ipv6-prefix
       +--:(destination-match)
          +--rw destination-match
             +--rw (match)?
                +--:(user-group)
                |  +--rw user-group-id?   uint32
                +--:(IP-address)
                   +--rw ipv4-network?    inet:ipv4-prefix
                   +--rw ipv6-network?    inet:ipv6-prefix
  augment /acl:acls/acl:acl/acl:aces/acl:ace:
    +--rw time-range {match-on-user-group}?
       +--rw (time-range-type)?
          +--:(periodic-range)
          |  +--rw month*          lmap:month-or-all
          |  +--rw day-of-month*   lmap:day-of-months-or-all
          |  +--rw day-of-week*    lmap:weekday-or-all
          |  +--rw hour*           lmap:hour-or-all
          +--:(absolute-range)
             +--rw start-time?     yang:date-and-time
             +--rw end-time?       yang:date-and-time
```

Figure 4: UCL Extension

This module specifies an extension to the IETF-ACL model [RFC8519] such that the UCL group index may be referenced by augmenting the "matches" node. Four types of UCL group are supported:

* U2U: Inter-groups communication, i.e., both source and destination identifiers are user groups.

* N2N: Both source and destination identifiers are IP address prefixes.

* U2N: The source identifier is one specific user group while the destination identifier is one specific IP address prefix.

* N2U: The source identifier is one specific IP address prefix while the destination identifier is one specific user group.

5.2.2. The YANG Module

This module imports [RFC6991], [RFC8194] and [RFC8519].

```
<CODE BEGINS>
 file="ietf-ucl-acl@2022-10-14.yang"
module ietf-ucl-acl {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ucl-acl";
  prefix uacl;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types, Section 3";
  }
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types, Section 4";
  }
  import ietf-lmap-common {
    prefix lmap;
    reference
      "RFC 8194: A YANG Data Model for LMAP Measurement Agents";
  }
  import ietf-access-control-list {
    prefix acl;
    reference
      "RFC 8519: YANG Data Model for Network Access
                 Control Lists (ACLs)";
  }

  organization
    "IETF OPSWG Working Group";
  contact
    "WG Web: <https://datatracker.ietf.org/wg/opsawg/>
     WG List: <mailto:opsawg@ietf.org>";
  description
    "This YANG module defines XXX.

     Copyright (c) 2022 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 Revised
     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 2022-10-14 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: A Policy-based Network Access Control";
    }

    feature match-on-user-group {
      description
        "The device can support matching on user groups.";
    }

    grouping match-range-source-destination {
      description
        "A grouping used for source/desttination macthes.";
      choice match {
        description
          "Add a new match choice for the user group.";
        case user-group {
          leaf user-group-id {
            type uint32;
            description
              "The matched user group ID that uniquely identifies
               a user group.";
          }
        }
        case IP-address {
          leaf ipv4-network {
            type inet:ipv4-prefix;
            description
              "The matched IPv4 address prefix.";
          }
          leaf ipv6-network {
            type inet:ipv6-prefix;
            description
              "The matched IPv6 address prefix.";
          }
        }
      }
    }
```

```
  augment "/acl:acls/acl:acl/acl:aces/acl:ace/acl:matches" {
    if-feature "match-on-user-group";
    description
      "Add new match types.";
    choice user-control-groups {
      description
        "Add new source and destination matches based on the
         user group.";
      container source-match {
        description
          "The source matched information.";
        uses match-range-source-destination;
      }
      container destination-match {
        description
          "The destination matched information.";
        uses match-range-source-destination;
      }
    }
  }

  augment "/acl:acls/acl:acl/acl:aces/acl:ace" {
    if-feature "match-on-user-group";
    description
      "Add a new parameter to the Access Control Entry (ACE).";
    container time-range {
      description
        "This container defines when the access control
         entry rules are in effect.

         If it is not configured, the access control entry
         is immediately and always in effect.";
      choice time-range-type {
        description
          "Choice based on the type of the time range.";
        case periodic-range {
          leaf-list month {
            type lmap:month-or-all;
            description
              "A set of months at which ace will trigger.
               The wildcard means all months.";
          }
          leaf-list day-of-month {
            type lmap:day-of-months-or-all;
            description
              "A set of days of the month at which ace will
               trigger. The wildcard means all days of a month.";
          }
```

```
            leaf-list day-of-week {
              type lmap:weekday-or-all;
              description
                "A set of weekdays at which ace will trigger.
                 The wildcard means all weekdays.";
            }
            leaf-list hour {
              type lmap:hour-or-all;
              description
                "A set of hours at which ace will trigger. The
                 wildcard means all hours of a day.";
            }
          }
          case absolute-range {
            leaf start-time {
              type yang:date-and-time;
              description
                "The time when the ace starts to take effect.";
            }
            leaf end-time {
              type yang:date-and-time;
              description
                "The time when the ace ends to take effect.";
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. User Access Control Group ID RADIUS Attribute

The User-Access-Group-ID RADIUS attribute and its embedded TLVs are defined with globally unique names. The definition of the attribute follows the guidelines in Section 2.7.1 of [RFC6929]. This attribute is used to indicate the user-group ID to be used by the UAD. When the User-Access-Group-ID RADIUS attribute is present in the RADIUS Access-Accept, the system applies the related access control to the users after it authenticates.

The value fields of the Attribute are encoded in clear and not encrypted as, for example, Tunnel- Password Attribute [RFC2868].

The User-Access-Group-ID Attribute is of type "string" as defined in Section 3.5 of [RFC8044].

The User-Access-Group-ID Attribute MAY appear in a RADIUS Access-Accept packet. It MAY also appear in a RADIUS Access-Request packet as a hint to the RADIUS server to indicate a preference. However, the server is not required to honor such a preference.

The User-Access-Group-ID Attribute MAY appear in a RADIUS CoA-Request packet.

The User-Access-Group-ID Attribute MAY appear in a RADIUS Accounting-Request packet.

The User-Access-Group-ID Attribute MUST NOT appear in any other RADIUS packet.

The User-Access-Group-ID Attribute is structured as follows:

Type

241

Length

This field indicates the total length, in octets, of all fields of this attribute, including the Type, Length, Extended-Type, and the "Value".

Extended-Type

TBA1

Value

This field contains the user group ID.

The User-Access-Group-ID Attribute is associated with the following identifier: 241.TBA1.

7. Table of Attributes

The following table provides a guide as what type of RADIUS packets that may contain User-Access-Group-ID Attribute, and in what quantity.

Access-Request	Access-Accept	Access-Reject	Challenge	Acct.-Request	#	Attribute
0+	0+	0	0	0+	241.TBA1	User-Access-Group-ID

CoA-Request	CoA-ACK	CoA-NACK	#	Attribute
0+	0	0	241.TBA2	User-Access-Group-ID

The following table defines the meaning of the above table entries:

0 This attribute MUST NOT be present in packet.
0+ Zero or more instances of this attribute MAY be present in packet.

8. Security Considerations

8.1. YANG

The YANG modules specified in this document defines 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 [RFC8446].

The Network Configuration Access Control Model (NACM) [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.

There are a number of data nodes defined in this YANG module that are writable, creatable, and deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations to these data nodes could have a negative effect on network and security operations.

<<add-more-about privacy considerations as the modules manipulate PII data.>>

8.2. RADIUS

RADIUS-related security considerations are discussed in [RFC2865].

This document targets deployments where a trusted relationship is in place between the RADIUS client and server with communication optionally secured by IPsec or Transport Layer Security (TLS) [RFC6614].

9. IANA Considerations

9.1. YANG

This document registers a URI in the "IETF XML Registry" [RFC3688]. Following the format in RFC 3688, the following registration has been made.

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

URI: urn:ietf:params:xml:ns:yang:ietf-ucl-acl
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-ucl-group
namespace:           urn:ietf:params:xml:ns:yang:ietf-ucl-group
prefix:              uclg
maintained by IANA: N
reference:           RFC XXXX

name:                ietf-ucl-acl
namespace:           urn:ietf:params:xml:ns:yang:ietf-ucl-acl
prefix:              uacl
maintained by IANA: N
reference:           RFC XXXX
```

9.2. RADIUS

IANA is requested to assign a new RADIUS attribute typs from the IANA registry "Radius Attribute Types" [RADIUS-Types]:

Value	Description	Data Type	Reference
241.TBA1	User-Access-Group-ID	string	This-Document

Table 1: Encrypted DNS RADIUS Attributes

10. Acknowledgements

This work has benefited from the discussions of User-group-based Security Policy over the years. In particular, [I-D.you-i2nsf-user-group-based-policy] and [I-D.yizhou-anima-ip-to-access-control-groups] provide mechanisms to establish the mapping between the IP address/prefix of user and access control group ID.

Jianjie You, Myo Zarny, Christian Jacquenet, Mohamed Boucadair, and Yizhou Li contributed to an earlier version of [I-D.you-i2nsf-user-group-based-policy]. We would like to thank the authors of that draft on modern network access control mechanisms for material that assisted in thinking about this document.

11. References

11.1. Informative References

[I-D.dbb-netmod-acl]
Dios, O. G. D., Barguil, S., and M. Boucadair, "Extensions to the Access Control Lists (ACLs) YANG Model", Work in Progress, Internet-Draft, draft-dbb-netmod-acl-01, 29 June 2022, <https://www.ietf.org/archive/id/draft-dbb-netmod-acl-01.txt>.

[NIST-ABAC]
Hu, Vincent C., "Guide to Attribute Based Access Control (ABAC) Definition and Considerations", January 2014.

[RADIUS-Types]
IANA, "RADIUS Types", <http://www.iana.org/assignments/radius-types>.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <https://www.rfc-editor.org/info/rfc2475>.

[RFC2868] Zorn, G., Leifer, D., Rubens, A., Shriver, J., Holdrege, M., and I. Goyret, "RADIUS Attributes for Tunnel Protocol Support", RFC 2868, DOI 10.17487/RFC2868, June 2000, <https://www.rfc-editor.org/info/rfc2868>.

[RFC3198] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, DOI 10.17487/RFC3198, November 2001, <https://www.rfc-editor.org/info/rfc3198>.

[RFC6614] Winter, S., McCauley, M., Venaas, S., and K. Wierenga, "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, DOI 10.17487/RFC6614, May 2012, <https://www.rfc-editor.org/info/rfc6614>.

11.2. 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>.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <https://www.rfc-editor.org/info/rfc2865>.

[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>.

[RFC6929] DeKok, A. and A. Lior, "Remote Authentication Dial In User Service (RADIUS) Protocol Extensions", RFC 6929, DOI 10.17487/RFC6929, April 2013, <https://www.rfc-editor.org/info/rfc6929>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[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>.

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

[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>.

[RFC8194] Schoenwaelder, J. and V. Bajpai, "A YANG Data Model for LMAP Measurement Agents", RFC 8194, DOI 10.17487/RFC8194, August 2017, <https://www.rfc-editor.org/info/rfc8194>.

[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>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

Authors' Addresses

Qiufang Ma
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: maqiufang1@huawei.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com

Mohamed Boucadair
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.com

Daniel King
Old Dog Consulting
United Kingdom
Email: daniel@olddog.co.uk