NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2024

A. Elhassany
Swisscom

1 March 2024

Validating anydata in YANG Library context
draft-aelhassany-anydata-validation-00

Abstract

This document describes a method to use yang-library to validate YANG data nodes with type anydata.

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 2 September 2024.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
2. Terminology 3
3. Validating "anydata" Data Tree 3
4. IANA Considerations 4
5. Security Considerations 4
6. References 4
6.1. Normative References 4
6.2. Informative References 4
Acknowledgements 5
Author's Address 5

1. Introduction

YANG [RFC7950] defines the "anydata" statement to represent an unknown set of YANG nodes for which the data model is not known at module design time. However, YANG [RFC7950] left the verification of the "anydata" tree open to be done using external means. Several IETF models, e.g., [RFC7895], [RFC8526], [RFC9144], [RFC8639], [RFC8641], and [RFC8040], use "anydata" in their definitions. Current YANG implementations accept syntactically valid YANG data nodes as children of an "anydata" node but do not check the semantics of these data nodes against a YANG schema. This creates a real problem for any consumer of these models when validating all leaves of the YANG data tree.

YANG Schema Mount [RFC8528] allows mounting complete data models at implementation and run time. While powerful, schema mount cannot address use cases where the user selects an arbitrary subset of an instantiated data tree, such as YANG PUSH [RFC8641]. A current proposed approach, YANG Full Include YANG Full Include [I-D.jouqui-netmod-yang-full-include], complements YANG Schema Mount and applies at design time, yet cannot address dynamic filtering of an instantiated YANG data tree.

In this document we propese using YANG Library [RFC7895] to define the context in which anydata trees are validated. This would require the YANG tooling to implement an optional flag that enables a a flag for validating "anydata" subtrees in the context of a YANG library.

1.1. Requirements Language

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

2. Terminology

We use the terminology defined in YANG [RFC7950] for schema node and schema tree but refine data node and data tree to be more precise.

* data node: A node in the schema tree that can be instantiated in a data tree. One of container, leaf, leaf-list, list, anydata, and anyxml. We do not discuss anyxml in this document, and this document does not change how YANG handles anyxml data nodes.

* instantiated data node: an instantiated instance of a data node that contains before fully qualified name (module namespace + identifier) for the data node and the data modeled within YANG.

* instantiated data tree: is what YANG [RFC7950] defines as "data tree". Adding the term "instantiated" precisely indicates that this tree is an instance of specific data modeled with YANG.

* data tree: a tree of data nodes (with no values).

3. Validating "anydata" Data Tree

The current YANG encodings, XML, JSON, and CBOR, encode instantiated data nodes with fully qualified name using the module’s namespace and a local name. The module’s namespace can be either explicit or assumed from a default namespace defined in the top data tree.

This document introduces a new YANG validation option: anydata-subtree-validation. In this mode, a YANG data parser MUST accept a YANG library as input along the YANG data file. When this option is enabled, any instantiated data node (NodeB) that is a child of anydata node (NodeA) is accepted to be valid only if (i) the qualified name of the node NodeB is found in one of the data trees defined by the YANG library AND (ii) the instantiated data tree rooted by NodeB is valid incomplete instantiated data tree according to the data node of NodeB.

The first condition ensures the completeness of the YANG library, and no subtree can be included as a child of anydata node unless a schema is defined for all the children of anydata subtree and specified in

the YANG library. The second condition applies a regular YANG validation against the subtree of anydata, considering that the subtree of anydata could be generated using an XPath [RFC8641] or a subtree filter [RFC6241]. Thus, the validator MUST consider this subtree incomplete and ignore any missing leaves.

4. IANA Considerations

This memo includes no request to IANA.

5. Security Considerations

TBD

6. References

6.1. 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>.

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

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

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

6.2. Informative References

[I-D.jouqui-netmod-yang-full-include] Joubert, T., Quilbeuf, J., and B. Claise, "YANG Full Include", Work in Progress, Internet-Draft, draft-jouqui-netmod-yang-full-include-00, 6 November 2023, <https://datatracker.ietf.org/doc/html/draft-jouqui-netmod-yang-full-include-00>.

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

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

[RFC8526] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "NETCONF Extensions to Support the Network Management Datastore Architecture", RFC 8526, DOI 10.17487/RFC8526, March 2019, <https://www.rfc-editor.org/info/rfc8526>.

[RFC8528] Bjorklund, M. and L. Lhotka, "YANG Schema Mount", RFC 8528, DOI 10.17487/RFC8528, March 2019, <https://www.rfc-editor.org/info/rfc8528>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

[RFC9144] Clemm, A., Qu, Y., Tantsura, J., and A. Bierman, "Comparison of Network Management Datastore Architecture (NMDA) Datastores", RFC 9144, DOI 10.17487/RFC9144, December 2021, <https://www.rfc-editor.org/info/rfc9144>.

Acknowledgements

The authors would like to thank Jean Quilbeuf and Thomas Graf for their review and valuable comments.

Author's Address

Ahmed Elhassany
Swisscom
Binzring 17
CH- Zuerich 8045
Switzerland
Email: ahmed.elhassany@swisscom.com

Delay-Tolerant Networking
Internet-Draft
Intended status: Informational
Expires: 31 August 2024

E.J. Birrane
S.E. Heiner
E. Annis
Johns Hopkins Applied Physics Laboratory

28 February 2024

DTN Management Architecture
draft-ietf-dtn-dtnma-12

Abstract

The Delay-Tolerant Networking (DTN) architecture describes a type of challenged network in which communications may be significantly affected by long signal propagation delays, frequent link disruptions, or both. The unique characteristics of this environment require a unique approach to network management that supports asynchronous transport, autonomous local control, and a small footprint (in both resources and dependencies) so as to deploy on constrained devices.

This document describes a DTN management architecture (DTNMA) suitable for managing devices in any challenged environment but, in particular, those communicating using the DTN Bundle Protocol (BP). Operating over BP requires an architecture that neither presumes synchronized transport behavior nor relies on query-response mechanisms. Implementations compliant with this DTNMA should expect to successfully operate in extremely challenging conditions, such as over uni-directional links and other places where BP is the preferred transport.

Table of Contents

1. Introduction 3
1.1. Scope 5
1.2. Organization 5
2. Terminology 6
3. Challenged Network Overview 8
3.1. Challenged Network Constraints 8
3.2. Topology and Service Implications 9
3.2.1. Management Implications 10
3.3. Management Special Cases 11
4. Desirable Design Properties 11
4.1. Dynamic Architectures 12
4.2. Hierarchically Modeled Information 12
4.3. Adaptive Push of Information 13
4.4. Efficient Data Encoding 14
4.5. Universal, Unique Data Identification 15
4.6. Runtime Data Definitions 16
4.7. Autonomous Operation 16
5. Current Remote Management Approaches 17
5.1. SNMP and SMI Models 18
5.1.1. The SMI Modeling Language 19
5.1.2. SNMP Protocol and Transport 19
5.2. XML-Infoset-Based Protocols and YANG Models 19
5.2.1. The YANG Modeling Language 20
5.2.2. NETCONF Protocol and Transport 22
5.2.3. RESTCONF Protocol and Transport 22
5.2.4. CORECONF Protocol and Transport 23
5.3. gRPC Network Management Interface (gNMI) 23
5.3.1. The Protobuf Modeling Language 23
5.3.2. gRPC Protocol and Transport 24
5.4. Intelligent Platform Management Interface (IPMI) 24
5.5. Autonomic Networking 24
5.6. Deep Space Autonomy 25

6. Motivation for New Features 25
7. Reference Model 26
7.1. Important Concepts 26
7.2. Model Overview 27
7.3. Functional Elements 28
7.3.1. Managed Applications and Services 28
7.3.2. DTNMA Agent (DA) 29
7.3.3. Managing Applications and Services 31
7.3.4. DTNMA Manager (DM) 32
7.3.5. Pre-Shared Definitions 34
8. Desired Services 34
8.1. Local Monitoring and Control 35
8.2. Local Data Fusion 35
8.3. Remote Configuration 36
8.4. Remote Reporting 37
8.5. Authorization 37
9. Logical Autonomy Model 38
9.1. Overview 38
9.2. Model Characteristics 40
9.3. Data Value Representation 42
9.4. Data Reporting 42
9.5. Command Execution 43
9.6. Predicate Autonomy Rules 44
10. Use Cases 45
10.1. Notation 45
10.2. Serialized Management 46
10.3. Intermittent Connectivity 47
10.4. Open-Loop Reporting 48
10.5. Multiple Administrative Domains 50
10.6. Cascading Management 52
11. IANA Considerations 54
12. Security Considerations 54
13. Informative References 54
Acknowledgements 60
Authors' Addresses 60

1. Introduction

The Delay-Tolerant Networking (DTN) architecture, as described in [RFC4838], has been designed to cope with data exchange in challenged networks. Just as the DTN architecture requires new capabilities for transport and transport security, special consideration is needed for the management of DTN devices.

This document describes a logical DTN Management Architecture (DTNMA) providing configuration, monitoring, and local control of both application and network services on a managed device. The DTNMA is designed to provide for the management of devices operating either within or across a challenged network.

> NOTE: A logical architecture describes the concepts and principles that support the logical operation of a system. This includes identifying components of the system (such as in a reference model), the behaviors they enable, and use cases describing how those behaviors result in the desired operation of the system. Logical architectures are not functional architectures.
>
> As such, this document does not specify a particular functional design for achieving desired behaviors. What is presented here is a set of architectural principles. It is not implied to be complete or to define interfaces between components.

Fundamental properties of a challenged network are outlined in Section 2.2.1 of [RFC7228]. These properties include lacking end-to-end IP connectivity, having "serious interruptions" to end-to-end connectivity, and exhibiting delays longer than can be tolerated by end-to-end synchronization mechanisms (such as TCP). It is further noted that the DTN architecture was designed to cope with such networks.

> NOTE: These challenges may be caused by a variety of factors such as physical constraints (e.g., long signal propagation delays and frequent link disruptions), administrative policies (e.g., quality-of-service prioritization, service-level agreements, and traffic management and scheduling), and off-nominal behaviors (e.g., active attackers and misconfigurations).

Device management in these environments occurs without human interactivity, without system-in-the-loop synchronous function, and without requiring a synchronous underlying transport layer. This means that managed devices need to determine their own schedules for data reporting, their own operational configuration, and perform their own error discovery and mitigation.

Certain outcomes of device self-management should be determinable by a privileged external observer (such as a managing device). In a challenged network, these observers may need to communicate with a managed device after significant periods of disconnectedness. Non-deterministic behavior of a managed device may make establishing communication difficult or impossible.

The desire to define asynchronous and autonomous device management is not new. However, challenged networks (in general) and the DTN environment (in particular) represent unique deployment scenarios and impose unique design constraints. To the extent that these environments differ from more traditional, enterprise networks, their management may also differ from the management of enterprise networks. Therefore, existing techniques may need to be adapted to operate in the DTN environment or new techniques may need to be created.

> NOTE: The DTNMA is designed to leverage any transport, network, and security solutions designed for challenged networks. However, the DTNMA specifically needs to be able to operate in any environment in which the Bundle Protocol (BPv7) [RFC9171] is deployed.

1.1. Scope

This document describes the desirable properties of, and motivation for, a DTNMA. This document also provides a reference model, service descriptions, autonomy model, and use cases to better reason about ways to standardize and implement this architecture.

This document provides informative guidance to authors and users of such models, protocols, and implementations.

The selection of any particular transport or network layer is outside of the scope of this document. The DTNMA does not require the use of any specific protocol such as IP, BP, TCP, or UDP. In particular, the DTNMA design does not assume the use of either IPv4 or IPv6.

> NOTE: The fact that the DTNMA can operate in any environment that deploys BP does not mean that the DTNMA requires the use of BP to operate.

Network features such as naming, addressing, routing, and communications security are out of scope of the DTNMA. It is presumed that any operational network communicating DTNMA messages would implement these services for any payloads carried by that network.

The interactions between and amongst the DTNMA and other management approaches are outside of the scope of this document.

1.2. Organization

The remainder of this document is organized into the following nine sections, described as follows.

Terminology: This section identifies terms fundamental to understanding DTNMA concepts. Whenever possible, these terms align in both word selection and meaning with their use in other management protocols.

Challenged Network Overview: This section describes important aspects of challenged networks and necessary approaches for their management.

Desirable Design Properties: This section defines those properties of the DTNMA needed to operate within the constraints of a challenged network. These properties are similar to the specification of system-level requirements of a DTN management solution.

Current Remote Management Approaches: This section provides a brief overview of existing remote management approaches. Where possible, the DTNMA adopts concepts from these approaches. The limitations of current approaches from the perspective of the DTNMA desirable properties are identified and discussed.

Motivation for New Features: This section provides an overall motivation for this work, to include explaining why a management architecture for challenged networks is useful and necessary.

Reference Model: This section defines a reference model that can be used to reason about the DTNMA independent of an implementation or implementation architecture. This model identifies the logical components of the system and the high-level relationships and behaviors amongst those components.

Desired Services: This section identifies and defines the DTNMA services provided to network and mission operators.

Logical Autonomy Model: This section provides an exemplar data model that can be used to reason about DTNMA control and data flows. This model is based on the DTNMA reference model.

Use Cases: This section presents multiple use cases accommodated by the DTNMA architecture. Each use case is presented as a set of control and data flows referencing the DTNMA reference model and logical autonomy model.

2. Terminology

This section defines terminology that either is unique to the DTNMA or is necessary for understanding the concepts defined in this specification.

Timely Data Exchange: The ability to communicate information between two (or more) entities within a required period of time. In some cases, a 1-second exchange may not be timely and in other cases 1-hour exchange may be timely.

DTN Management: Management that does not depend on stateful connections, timely data exchange of management messages, or system-in-the-loop synchronous functions.

DTNMA Agent (DA): A role associated with a managed device, responsible for reporting performance data, accepting policy directives, performing autonomous local control, error-handling, and data validation. DAs exchange information with DMs operating either on the same device and/or on remote devices in the network.

DTNMA Manager (DM): A role associated with a managing device responsible for configuring the behavior of, and eventually receiving information from, DAs. DMs interact with one or more DAs located on the same device and/or on remote devices in the network.

Controls: Procedures run by a DA to change the behavior, configuration, or state of an application or protocol managed by that DA. This includes procedures to manage the DA itself, such as to have the DA produce performance reports or to apply new management policies.

Externally Defined Data (EDD): Typed information made available to a DA by its hosting device, but not computed directly by the DA itself.

Data Reports: Typed, ordered collections of data values gathered by one or more DAs and provided to one or more DMs. Reports comply to the format of a given Data Report Schema.

Data Report Schemas: Named, ordered collection of data elements that represent the schema of a Data Report.

Rules: Unit of autonomous specification that provides a stimulus-response relationship between time or state on a DA and the actions or operations to be run as a result of that time or state.

3. Challenged Network Overview

The DTNMA provides network management services able to operate in a challenged network environment, such as envisioned by the DTN architecture. This section describes what is meant by the term "challenged network", the important properties of such a network, and observations on impacts to conventional management approaches.

3.1. Challenged Network Constraints

Constrained networks are defined as networks where "some of the characteristics pretty much taken for granted with link layers in common use in the Internet at the time of writing are not attainable." [RFC7228]. This broad definition captures a variety of potential issues relating to physical, technical, and regulatory constraints on message transmission. Constrained networks typically include nodes that regularly reboot or are otherwise turned off for long periods of time, transmit at low or asynchronous bitrates, and/or have very limited computational resources.

Separately, a challenged network is defined as one that "has serious trouble maintaining what an application would today expect of the end-to-end IP model" [RFC7228]. Links in such networks may be impacted by attenuation, propagation delays, mobility, occultation, and other limitations imposed by energy and mass considerations. Therefore, systems relying on such links cannot guarantee timely end-to-end data exchange.

> NOTE: Because challenged networks might not provide services expected of the end-to-end IP model, devices in such networks might not implement networking stacks associated with the end-to-end IP model. This means that devices might not include support for certain transport protocols (TCP/UDP), web protocols (HTTP), or internetworking protocols (IPv4/IPv6).

By these definitions, a "challenged" network is a special type of "constrained" network, where constraints prevent timely end-to-end data exchange. As such, "all challenged networks are constrained networks ... but not all constrained networks are challenged networks ... Delay-Tolerant Networking (DTN) has been designed to cope with challenged networks" [RFC7228].

Solutions that work in constrained networks might not be solutions that work in challenged networks. In particular, challenged networks exhibit the following properties that impact the way in which the function of network management is considered.

* Timely end-to-end data exchange cannot be guaranteed to exist at any given time between any two nodes.

* Latencies on the order of seconds, hours, or days must be tolerated.

* Individual links may be uni-directional.

* Bi-directional links may have asymmetric data rates.

* The existence of external infrastructure, services, systems, or processes such as a Domain Name Service (DNS) or a Certificate Authority (CA) cannot be guaranteed.

3.2. Topology and Service Implications

The set of constraints that might be present in a challenged network impact both the topology of the network and the services active within that network.

Operational networks handle cases where nodes join and leave the network over time. These topology changes may or may not be planned, they may or may not represent errors, and they may or may not impact network services. Challenged networks differ from other networks not in the present of topological change, but in the likelihood that impacts to topology result in impacts to network services.

The difference between topology impacts and service impacts can be expressed in terms of connectivity. Topological connectivity usually refers to the existence of a path between an application message source and destination. Service connectivity, alternatively, refers to the existence of a path between a node and one or more services needed to process (often just-in-time) application messaging. Examples of service connectivity include access to infrastructure services such as a Domain Name System (DNS) or a Certificate Authority (CA).

In networks that might be partitioned most of the time, it is less likely that a node would concurrently access both an application endpoint and one or more network service endpoints. For this reason, network services in a challenged network should be designed to allow for asynchronous operation. Accommodating this use case often involves the use of local caching, pre-placing information, and not hard-coding message information at a source that might change when a message reaches its destination.

> NOTE: One example of rethinking services in a challenged network is the securing of BPv7 bundles. The BPSec [RFC9172] security extensions to BPv7 do not encode security destinations when applying security. Instead, BPSec requires nodes in a network to identify themselves as security verifiers or acceptors when receiving and processing secured messages.

3.2.1. Management Implications

Network management approaches need to adapt to the topology and service impacts encountered in challenged networks. In particular, the ways in which "managers" and "agents" operate will need to adapt to changes in connectivity and service endpoints.

When connectivity to a manager cannot be guaranteed, agents will need to rely on locally available information and use local autonomy to react to changes at the node. Architectures that rely on external resources such as access to third-party oracles, operators-in-the-loop, or other service infrastructure may fail to operate in a challenged network.

In addition to disconnectivity, topological change can alter the associations amongst managed and managing devices. Different managing devices might be active in a network at different times or in different partitions. Managed devices might communicate with some, all, or none of these managing devices as a function of their own local configuration and policy.

> NOTE: These concepts relate to practices in conventional networks. For example, supporting multiple managing devices is similar to deploying multiple instances of a network service -- such as a DNS server or CA node. Selecting from a set of managing devices is similar to a sensor node practice of electing cluster heads to act as privileged nodes for data storage and exfiltration.

Therefore, a network management architecture for challenged networks should:

1. Support a many-to-many association amongst managing and managed devices, and

2. Allow "control from" and "reporting to" managing devices to function independent of one another.

3.3. Management Special Cases

The following special cases illustrate some of the operational situations that can be encountered in the management of devices in a challenged network.

One-Way Management: A managed device can only be accessed via a uni-directional link, or a via a link whose duration is shorter than a single round-trip propagation time.

Summary Data: A managing device can only receive summary data of a managed device's state because a link or path is constrained by capacity or reliability.

Bulk Historical Reporting: A managing device receives a large volume of historical report data for a managed device. This can occur when a managed device rejoins a network or has access to a high capacity link (or path) to the managed device.

Multiple Managers A managed device tracks multiple managers in the network and communicates with them as a function of time, local state, or network topology. This includes challenged networks that interconnect two or more unchallenged networks such that managed and managing devices exist in different networks.

These special cases highlight the need for managed devices to operate without presupposing a dedicated connection to a single managing device. To support this, managing devices deliver instruction sets that govern the local, autonomous behavior of managed devices. These behaviors include (but are not limited to) collecting performance data, state, and error conditions, and applying pre-determined responses to pre-determined events. Managing devices in a challenged network might never expect a reply to a command, and communications from managed devices may be delivered much later than the events being reported.

4. Desirable Design Properties

This section describes those design properties that are desirable when defining a management architecture operating across challenged links in a network. These properties ensure that network management capabilities are retained even as delays and disruptions in the network scale. Ultimately, these properties are the driving design principles for the DTNMA.

> NOTE: These properties may influence the design, construction, and adaptation of existing management tools for use in challenged networks. For example, the properties the DTN

> architecture [RFC4838] resulted in the development of BPv7 [RFC9171] and BPSec [RFC9172]. The DTNMA may result in the construction of new management data models, policy expressions, and/or protocols.

4.1. Dynamic Architectures

The DTNMA should be agnostic of the underlying physical topology, transport protocols, security solutions, and supporting infrastructure of a given network. Due to the likelihood of operating in a frequently partitioned environment, the topology of a network may change over time. Attempts to stabilize an architecture around individual nodes can result in a brittle management framework and the creation of congestion points during periods of connectivity.

The DTNMA should not prescribe any association between a DM and a DA other than those defined in this document. There should be no logical limitation to the number of DMs that can control a DA, the number of DMs that a DA should report to, or any requirement that a DM and DA relationship implies a pair.

> NOTE: Practical limitations on the relationships between and amongst DMs and DAs will exist as a function of the capabilities of networked devices. These limitations derive from processing and storage constraints, performance requirements, and other engineering factors. While this information is vital to the proper engineering of a managed and managing device, they are implementation considerations, and not otherwise design constraints on the DTNMA.

4.2. Hierarchically Modeled Information

The DTNMA should use data models to define the syntactic and semantic contracts for data exchange between a DA and a DM. A given model should have the ability to "inherit" the contents of other models to form hierarchical data relationships.

> NOTE: The term data model in this context refers to a schema that defines a contract between a DA and a DM for how information is represented and validated.

Many network management solutions use data models to specify the semantic and syntactic representation of data exchanged between managed and managing devices. The DTNMA is not different in this regard - information exchanged between DAs and DMs should conform to one or more pre-defined, normative data models.

A common best practice when defining a data model is to make it cohesive. A cohesive model is one that includes information related to a single purpose such as managing a single application or protocol. When applying this practice, it is not uncommon to develop a large number of small data models that, together, describe the information needed to manage a device.

Another best practice for data model development is the use of inclusion mechanisms to allow one data model to include information from another data model. This ability to include a data model avoids repeating information in different data models. When one data model includes information from another data model, there is an implied model hierarchy.

Data models in the DTNMA should allow for the construction of both cohesive models and hierarchically related models. These data models should be used to define all sources of information that can be retrieved, configured, or executed in the DTNMA. This includes supporting DA autonomy functions such as parameterization, filtering, and event driven behaviors. These models will be used to both implement interoperable autonomy engines on DAs and define interoperable report parsing mechanisms on DMs.

> NOTE: While data model hierarchies can result in a more concise data model, arbitrarily complex nesting schemes can also result in very verbose encodings. Where possible, data identifications schemes should be constructed that allow for both hierarchical data and highly compressible data identification.

4.3. Adaptive Push of Information

DAs in the DTNMA architecture should determine when to push information to DMs as a function of their local state.

Pull management mechanisms require a managing device to send a query to a managed device and then wait for a response to that specific query. This practice implies some serialization mechanism (such as a control session) between entities. However, challenged networks cannot guarantee timely round-trip data exchange. For this reason, pull mechanisms should be avoided in the DTNMA.

Push mechanisms, in this context, refer to the ability of DAs to leverage local autonomy to determine when and what information should be sent to which DMs. The push is considered adaptive because a DA determines what information to push (and when) as an adaptation to changes to the DA's internal state. Once pushed, information might still be queued pending connectivity of the DA to the network.

> NOTE: Even in cases where a round-trip exchange can occur, pull mechanisms increase the overall amount of traffic in the network and preclude the use of autonomy at managed devices. So even when pull mechanisms are feasible they should not be considered a pragmatic alternative to push mechanisms.

4.4. Efficient Data Encoding

Messages exchanged between a DA and a DM in the DTNMA should be defined in a way that allows for efficient on-the-wire encoding. DTNMA design decisions that result in smaller message sizes should be preferred over those that result in larger message sizes.

There is a relationship between message encoding and message processing time at a node. Messages with little or no encodings may simplify node processing whereas more compact encodings may require additional activities to generate/parse encoded messages. Generally, compressing a message takes processing time at the sender and decompressing a message takes processing time at a receiver. Therefore, there is a design tradeoff between minimizing message sizes and minimizing node processing.

> NOTE: There are many ways in which message size, number of messages, and node behaviors can impact processing performance. Because the DTNMA does not presuppose any underlying protocol or implementation, this section is focused solely on the compactness of an individual message and the processing for encoding and decoding that individual message.

However, there is a significant advantage to smaller message sizes in a challenged network. Smaller messages require smaller periods of viable transmission for communication, they incur less re-transmission cost, and they consume less resources when persistently stored en-route in the network.

> NOTE: Naive approaches to minimizing message size through general purpose compression algorithms do not produce minimal encodings. Data models can, and should, be designed for compact encoding from the beginning. Design strategies for compact encodings involve using structured data, hierarchical data models, and common structures in data models. These strategies allow for compressibility beyond what would otherwise be achieved by computing large hash values over generalized data structures.

4.5. Universal, Unique Data Identification

Data elements within the DTNMA should be uniquely identifiable so that they can be individually manipulated. Further, these identifiers should be universal - the identifier for a data element should be the same regardless of role, implementation, or network instance.

Identification schemes that are relative to a specific DA or specific system configuration might change over time. In particular, nodes in a challenged network may change their status or configuration during periods of partition from other parts of the network. Resynchronizing relative state or configuration should be avoided whenever possible.

> NOTE: Consider the common technique for approximating an associative array lookup. For example, if a managed device tracks the number of bytes passed by multiple named interfaces, then the number of bytes through a specific named interface (say, "int_foo"), would be retrieved in the following way:
>
> 1. Query a list of ordered interface names from an agent.
> 2. Find the name that matches "int_foo" and infer the agent's index of "int_foo" from the ordered interface list. In this instance, say "int_foo" is the 4th interface in the list.
> 3. Query the agent to return the number of bytes passed through the 4th interface.
>
> Ignoring the inefficiency of two round-trip exchanges, this mechanism will fail if the agent changes its index mapping between the first and second query. For example, were "int_foo" to be restarted and slotted in a different index position. While this is unlikely to occur in a low-latency network, it is more likely to occur in a challenged network.
>
> The desired data being queried, "number of bytes through int_foo" should be uniquely and universally identifiable and independent of how that data exists in an agent's custom implementation.

4.6. Runtime Data Definitions

The DTNMA allows for the addition of new data elements to a data model as part of the runtime operation of the management system. These definitions may represent custom data definitions that are applicable only for a particular device or network. Custom definitions should also be able to be removed from the system during runtime.

The goal of this approach is to dynamically add or remove data elements to the local runtime schemas as needed - such as the definition of new counters, new reports, or new rules.

The custom definition of new data from existing data (such as through data fusion, averaging, sampling, or other mechanisms) provides the ability to communicate desired information in as compact a form as possible.

> NOTE: A DM could, for example, define a custom data report that includes only summary information around a specific operational event or as part of specific debugging. DAs could then produce this smaller report until it is no longer necessary, at which point the custom report could be removed from the management system.

Custom data elements should be calculated and used both as parameters for DA autonomy and for more efficient reporting to DMs. Defining new data elements allows for DAs to perform local data fusion and defining new reporting templates allows for DMs to specify desired formats and generally save on link capacity, storage, and processing time.

4.7. Autonomous Operation

The management of applications by a DA should be achievable using only knowledge local to the DA because DAs might need to operate during times when they are disconnected from a DM.

DA autonomy may be used for simple automation of predefined tasks or to support semi-autonomous behavior in determining when to run tasks and how to configure or parameterize tasks when they are run.

Important features provided by the DA are listed below. These features work together to accomplish tasks. As such, there is commonality amongst their definitions and nature of their benefits.

Stand-alone Operation: Pre-configuration allows DAs to operate

without regular contact with other nodes in the network. Updates for configurations remain difficult in a challenged network, but this approach removes the requirement that a DM be in-the-loop during regular operations. Sending stimuli-and-responses to a DA during periods of connectivity allows DAs to self-manage during periods of disconnectivity.

Deterministic Behavior: Operational systems might need to act in a deterministic way even in the absence of an operator in-the-loop. Deterministic behavior allows an out-of-contact DM to predict the state of a DA and to determine how a DA got into a particular state.

Engine-Based Behavior: Operational systems might not be able to deploy "mobile code" [RFC4949] solutions due to network bandwidth, memory or processor loading, or security concerns. Engine-based approaches provide configurable behavior without incurring these concerns.

Authorization, and Accounting: The DTNMA does not require a specific underlying transport protocol, network infrastructure, or network services. Therefore, mechanisms for authorization and accounting need to be present in a standard way at DAs and DMs to provide these functions if the underlying network does not. This is particularly true in cases where multiple DMs may be active concurrently in the network.

To understand the contributions of these features to a common behavior, consider the example of a managed device coming online with a set of pre-installed configuration. In this case, the device's stand-alone operation comes from the pre-configuration of its local autonomy engine. This engine-based behavior allows the system to behave in a deterministic way and any new configurations will need to be authorized before being adopted.

Features such as deterministic processing and engine-based behavior do not preclude the use of other Artificial Intelligence (AI) and Machine Learning (ML) approaches on a managed device.

5. Current Remote Management Approaches

Several remote management solutions have been developed for both local-area and wide-area networks. Their capabilities range from simple configuration and report generation to complex modeling of device settings, state, and behavior. Each of these approaches are successful in the domains for which they have been built, but are not all equally functional when deployed in a challenged network.

Remote management tools designed for unchallenged networks provide synchronous mechanisms for communicating locally-collected data from devices to operators. Applications are typically managed using a "pull" mechanism, requiring a managing device to explicitly request the data to be produced and transmitted by a managed device.

> NOTE: Network management solutions that pull large sets of data might not operate in a challenged environment that cannot support timely round-trip exchange of large data volumes.

More recent network management tools focus on message-based management, reduced state keeping by managed and managing devices, and increased levels of system autonomy.

This section describes some of the well-known protocols for remote management and contrasts their purposes with the desirable properties of the DTNMA. The purpose of this comparison is to identify parts of existing approaches that can be adopted or adapted for use in challenged networks and where new capabilities should be created specifically for this environment.

5.1. SNMP and SMI Models

An early and widely used example of a remote management protocol is the Simple Network Management Protocol (SNMP) currently at Version 3 [RFC3410]. The SNMP utilizes a request/response model to get and set data values within an arbitrarily deep object hierarchy. Objects are used to identify data such as host identifiers, link utilization metrics, error rates, and counters between application software on managing and managed devices [RFC3411]. Additionally, SNMP supports a model for unidirectional push messages, called event notifications, based on agent-defined triggering events.

SNMP relies on logical sessions with predictable round-trip latency to support its "pull" mechanism but a single activity is likely to require many round-trip exchanges. Complex management can be achieved, but only through careful orchestration of real-time, end-to-end, managing-device-generated query-and-response logic.

There is existing work that uses the SNMP data model to support some low-fidelity Agent-side processing, to include the Distributed Management Expression MIB [RFC2982] and Definitions of Managed Objects for the Delegation of Management Scripts [RFC3165]. However, Agent autonomy is not an SNMP mechanism, so support for a local agent response to an initiating event is limited. In a challenged network where the delay between a managing device receiving an alert and sending a response can be significant, SNMP is insufficient for autonomous event handling.

5.1.1. The SMI Modeling Language

SNMP separates the representations for managed data models from Manager--Agent messaging, sequencing and encoding. Each data model is termed a Management Information Base (MIB) [RFC3418] and uses the Structure of Management Information (SMI) modeling language [RFC2578]. Additionally, the SMI itself is based on the ASN.1 Syntax [ASN.1] which is used not just for SMI but for other, unrelated data structure specification such as the Cryptographic Message Syntax (CMS) [RFC5652]. Separating data models from messaging and encoding is a best practice in remote management protocols and is also necessary for the DTNMA.

Each SNMP MIB is composed of managed object definitions each of which is associated with a hierarchical Object Identifier (OID). Because of the arbitrarily deep nature of MIB object trees, the size of OIDs is not strictly bounded by the protocol (though may be bounded by implementations).

5.1.2. SNMP Protocol and Transport

The SNMP protocol itself, which is at version 2 [RFC3416], can operate over a variety of transports, including plaintext UDP/IP [RFC3417], SSH/TCP/IP [RFC5592], and DTLS/UDP/IP or TLS/TCP/IP [RFC5953].

SNMP uses an abstracted security model to provide authentication, integrity, and confidentiality. There are options for user-based security model (USM) of [RFC3414], which uses in-message security, and transport security model (TSM) [RFC5591], which relies on the transport to provide security functions and interfaces.

5.2. XML-Infoset-Based Protocols and YANG Models

Several network management protocols, including NETCONF [RFC6241], RESTCONF [RFC8040], and CORECONF [I-D.ietf-core-comi], share the same XML information set [xml-infoset] for its hierarchical managed information and [XPath] expressions to identify nodes of that information model. Since they share the same information model and the same data manipulation operations, together they will be referred to as "*CONF" protocols. Each protocol, however, provides a different encoding of that information set and its related operation-specific data.

The YANG modeling language of [RFC7950] is used to define the data model for these management protocols. Currently, YANG represents the IETF standard for defining managed information models.

5.2.1. The YANG Modeling Language

The YANG modeling language defines a syntax and modular semantics for organizing and accessing a device's configuration or operational information. YANG allows subdividing a full managed configuration into separate namespaces defined by separate YANG modules. Once a module is developed, it is used (directly or indirectly) on both the client and server to serve as a contract between the two. A YANG module can be complex, describing a deeply nested and inter-related set of data nodes, actions, and notifications.

Unlike the separation in Section 5.1.1 between ASN.1 syntax and module semantics from higher-level SMI data model semantics, YANG defines both a text syntax and module semantics together with data model semantics.

The YANG language provides flexibility in the organization of model objects to the model developer. The YANG supports a broad range of data types noted in [RFC6991]. YANG supports the definition of parameterized Remote Procedure Calls (RPCs) and actions to be executed on managed devices as well as the definition of event notifications within the model.

> Current *CONF notification logic allows a client to subscribe to the delivery of specific containers or data nodes defined in the model, either on a periodic or "on change" basis [RFC8641]. These notification events can be filtered according to XPath [XPath] or subtree [RFC6241] filtering as described in Section 2.2 of [RFC8639].

The use of YANG for data modeling necessarily comes with some side-effects, some of which are described here.

Text Naming: Data nodes, RPCs, and notifications within a YANG model are named by a namespace-qualified, text-based path of the module, sub-module, container, and any data nodes such as lists, leaf-lists, or leaves, without any explicit hierarchical organization based on data or object type.

Existing efforts to make compressed names for YANG objects, such as the YANG Schema Item iDentifiers (SID) from Section 3.2 of [RFC9254], allow a node to be named by an globally unique integer value but are still relatively verbose (up to 8 bytes per item) and still must be translated into text form for things like instance identification (see below). Additionally, when representing a tree of named instances the child elements can use differential encoding of SID integer values as "delta" integers. The mechanisms for assigning SIDs and the lifecycle of those SIDs are still in development [I-D.ietf-core-sid].

Text Values and Built-In Types: Because the original use of YANG with NETCONF was to model XML information sets, the values and built-in types are necessarily text based. The JSON encoding of YANG data [RFC7951] allows for optimized representations of many built-in types, and similarly the CBOR encoding [RFC9254] allows for different optimized representations.

In particular, the YANG built-in types natively support a fixed range of decimal fractions (Section 9.3 of [RFC7950]) but purposefully do not support floating point numbers. There are alternatives, such as the type bandwidth-ieee-float32 from [RFC8294] or using the "binary" type with one of the IEEE-754 encodings.

Deep Hierarchy: YANG allows for, and current YANG modules take advantage of, the ability to deeply nest a model hierarchy to represent complex combinations and compositions of data nodes. When a model uses a deep hierarchy of nodes this necessarily means that the qualified paths to name those nodes and instances is longer than a flat hierarchy would be.

Instance Identification: The node instances in a YANG module necessarily use XPath expressions for identification. Some identification is constrained to be strictly within the YANG domain, such as "must" "when", "augment", or "deviation" statements. Other identification needs to be processed by a managed device, such as in "instance-identifier" built-in type. This means any implementation of a managed device must include XPath processing and related information model handling of Section 6.4 of [RFC7950] and its referenced documents.

Protocol Coupling: A significant amount of existing YANG tooling or modeling presumes the use of YANG data within a management protocol with specific operations available. For exmaple, the access control model of [RFC8341] relies on those operations specific to the *CONF protocols for proper behavior.

The emergence of multiple NETCONF-derived protocols may make these presumptions less problematic in the future. Work to more consistently identify different types of YANG modules and their use has been undertaken to disambiguate how YANG modules should be treated [RFC8199].

Manager-Side Control: YANG RPCs and actions execute on a managed device and generate an expected, structured response. RPC execution is strictly limited to those issued by the manager. Commands are executed immediately and sequentially as they are received by the managed device, and there is no method to autonomously execute RPCs triggered by specific events or conditions.

The YANG modeling language continues to evolve as new features are needed by adopting management protocols.

5.2.2. NETCONF Protocol and Transport

NETCONF is a stateful, XML-encoding-based protocol that provides a syntax to retrieve, edit, copy, or delete any data nodes or exposed functionality on a server. It requires that underlying transport protocols support long-lived, reliable, low-latency, sequenced data delivery sessions. A bi-directional NETCONF session needs to be established before any data transfer (or notification) can occur.

The XML exchanged within NETCONF messages is structured according to YANG modules supported by the NETCONF agent, and the data nodes reside within one of possibly many datastores in accordance with the Network Management Datastore Architecture (NMDA) of [RFC8342].

NETCONF transports are required to provide authentication, data integrity, confidentiality, and replay protection. Currently, NETCONF can operate over SSH/TCP/IP [RFC6242] or TLS/TCP/IP [RFC7589].

5.2.3. RESTCONF Protocol and Transport

RESTCONF is a stateless, JSON-encoding-based protocol that provides the same operations as NETCONF, using the same YANG modules for structure and same NMDA datastores, but using RESTful exchanges over HTTP. It uses HTTP-native methods to express its allowed operations: GET, POST, PUT, PATCH, or DELETE data nodes within a datastore.

Although RESTCONF is a logically stateless protocol, it does rely on state within its transport protocol to achieve behaviors such as authentication and security sessions. Because RESTCONF uses the same data node semantics of NETCONF, a typical activity can involve the use of several sequential round-trips of exchanges to first discover managed device state and then act upon it.

5.2.4. CORECONF Protocol and Transport

CORECONF is an emerging stateless protocol built atop the Constrained Application Protocol (CoAP) [RFC7252] that defines a messaging construct developed to operate specifically on constrained devices and networks by limiting message size and fragmentation. CoAP also implements a request/response system and methods for GET, POST, PUT, and DELETE.

5.3. gRPC Network Management Interface (gNMI)

Another emerging but not-IETF-affiliated management protocol is the gRPC Network Management Interface (gNMI) [gNMI] which is based on gRPC messaging and uses Protobuf data modeling.

The same limitations of RESTCONF listed above apply to gNMI because of its reliance on synchronous HTTP exchanges and TLS security for normal operations, as well as the likely deep nesting of data schemas. There is a capability for gNMI to transport JSON-encoded YANG-modeled data, but this composing is not fully standardized and relies on specific tool integrations to operate.

5.3.1. The Protobuf Modeling Language

The data managed and exchanged via gNMI is encoded and modeled using Google Protobuf, an encoding and modeling syntax not affiliated with the IETF (although an attempt has been made and abandoned [I-D.rfernando-protocol-buffers]).

Because the Protobuf modeling syntax is relatively low-level (around the same as ASN.1 or CBOR), there are some efforts as part of the OpenConfig work [gNMI] to translate YANG modules into Protobuf schemas (similar to translation to XML or JSON schemas for NETCONF and RESTCONF respectively) but there is no required interoperabilty between management via gRPC or any of the *CONF protocols.

5.3.2. gRPC Protocol and Transport

The message encoding and exchange for gNMI, as the name implies, is gRPC protocol [gRPC]. gRPC exclusively uses HTTP/2 [RFC7540] for transport and relies on some aspects specific to HTTP/2 for its operations (such as HTTP trailer fields). While not mandated by gRPC, when used to transport gNMI data TLS is required for transport security.

5.4. Intelligent Platform Management Interface (IPMI)

A lower-level remote management protocol, intended to be used to manage hardware devices and network appliances below the operating system (OS), is the Intelligent Platform Management Interface (IPMI) standardized in [IPMI]. The IPMI is focused on health monitoring, event logging, firmware management, and serial-over-LAN (SOL) remote console access in a "pre-OS or OS-absent" host environment. The IPMI operates over a companion Remote Management Control Protocol (RMCP) for messaging, which itself can use UDP for transport.

Because the IPMI and RCMP are tailored to low-level and well-connected devices within a datacenter, with typical workflows requiring many messaging round trips or low-latency interactive sessions, they are not suitable for operation over a challenged network.

5.5. Autonomic Networking

The future of network operations requires more autonomous behavior including self-configuration, self-management, self-healing, and self-optimization. One approach to support this is termed Autonomic Networking [RFC7575].

There is a large and growing set of work within the IETF focused on developing an Autonomic Networking Integrated Model and Approach (ANIMA). The ANIMA work has developed a comprehensive reference model for distributing autonomic functions across multiple nodes in an autonomic networking infrastructure [RFC8993].

This work, focused on learning the behavior of distributed systems to predict future events, is an emerging network management capability. This includes the development of signalling protocols such as GRASP [RFC8990] and the autonomic control plane (ACP) [RFC8368].

Both autonomic and challenged networks require similar degrees of autonomy. However, challenged networks cannot provide the complex coordination between nodes and distributed supporting infrastructure necessary for the frequent data exchanges for negotiation, learning, and bootstrapping associated with the above capabilities.

There is some emerging work in ANIMA as to how disconnected devices might join and leave the autonomic control plane over time. However, this work is solving an important, but different, problem than that encountered by challenged networks.

5.6. Deep Space Autonomy

Outside of the terrestrial networking community, there are existing and established remote management systems used for deep space mission operations. Examples of two of these are for the New Horizons mission to Pluto [NEW-HORIZONS] and the DART mission to the asteroid Dimorphos [DART].

The DTNMA has some heritage in the concepts of deep space autonomy, but each of those mission instantiations use mission-specific data encoding, messaging, and transport as well as mission-specific (or heavily mission-tailored) modeling concepts and languages. Part of the goal of the DTNMA is to take the proven concepts from these missions and standardize the messaging syntax as well as a modular data modeling method.

6. Motivation for New Features

The future of network management will involve autonomous and autonomic functions operating on both managed and managing devices. However, the development of distributed autonomy for coordinated learning and event reaction is different from a managed device operating without connectivity to a managing node.

Management mechanisms that provide DTNMA desirable properties do not currently exist. This is not surprising since autonomous management in the context of a challenged networking environment is an emerging use case.

In particular, a management architecture is needed that provides the following new features.

Open Loop Control: Freedom from a request-response architecture, API, or other presumption of timely round-trip communications. This is particularly important when managing networks that are not built over an HTTP or TCP/TLS infrastructure.

Standard Autonomy Model: An autonomy model that allows for standard expressions of policy to guarantee deterministic behavior across devices and vendor implementations.

Compressible Model Structure: A data model that allows for very compact encodings by defining and exploiting common structures for data schemas.

Combining these new features with existing mechanisms for message data exchange (such as BP), data representations (such as CBOR) and data modeling languages (such as YANG) will form a pragmatic approach to defining challenged network management.

7. Reference Model

There are a multitude of ways in which both existing and emerging network management protocols, APIs, and applications can be integrated for use in challenged environments. However, expressing the needed behaviors of the DTNMA in the context of any of these pre-existing components risks conflating systems requirements, operational assumptions, and implementation design constraints.

7.1. Important Concepts

This section describes a network management concept for challenged networks (generally) and those conforming to the DTN architecture (in particular). The goal of this section is to describe how DTNMA services provide DTNMA desirable properties.

> NOTE: This section assumes a BPv7 underlying network transport. Bundles are the baseline transport protocol data units of the DTN architecture. Additionally, they may be used in a variety of network architectures beyond the DTN architecture. Therefore, assuming bundles is a convenient way of scoping DTNMA to any network or network architecture that relies on BPv7 features.

Similar to other network management architectures, the DTNMA draws a logical distinction between a managed device and a managing device. Managed devices use a DA to manage resident applications. Managing devices use a DM to both monitor and control DAs.

> NOTE: The terms "managing" and "managed" represent logical characteristics of a device and are not, themselves, mutually exclusive. For example, a managed device might, itself, also manage some other device in the network. Therefore, a device may support either or both of these characteristics.

The DTNMA differs from some other management architectures in three significant ways, all related to the need for a device to self-manage when disconnected from a managing device.

Pre-shared Definitions: Managing and managed devices should operate using pre-shared data definitions and models. This implies that static definitions should be standardized whenever possible and that managing and managed devices may need to negotiate definitions during periods of connectivity.

Agent Self-Management: A managed device may find itself disconnected from its managing device. In many challenged networking scenarios, a managed device may spend the majority of its time without a regular connection to a managing device. In these cases, DAs manage themselves by applying pre-shared policies received from managing devices.

Command-Based Interface: Managing devices communicate with managed devices through a command-based interface. Instead of exchanging variables, objects, or documents, a managing device issues commands to be run by a managed device. These commands may create or update variables, change data stores, or impact the managed device in ways similar to other network management approaches. The use of commands is, in part, driven by the need for DAs to receive updates from both remote management devices and local autonomy. The use of controls for the implementation of commands is discussed in more detail in Section 9.5.

7.2. Model Overview

A DTNMA reference model is provided in Figure 1 below. In this reference model, applications and services on a managing device communicate with a DM which uses pre-shared definitions to create a set of policy directives that can be sent to a managed device's DA via a command-based interface. The DA provides local monitoring and control (commanding) of the applications and services resident on the managed device. The DA also performs local data fusion as necessary to synthesize data products (such as reports) that can be sent back to the DM when appropriate.

DTNMA Reference Model

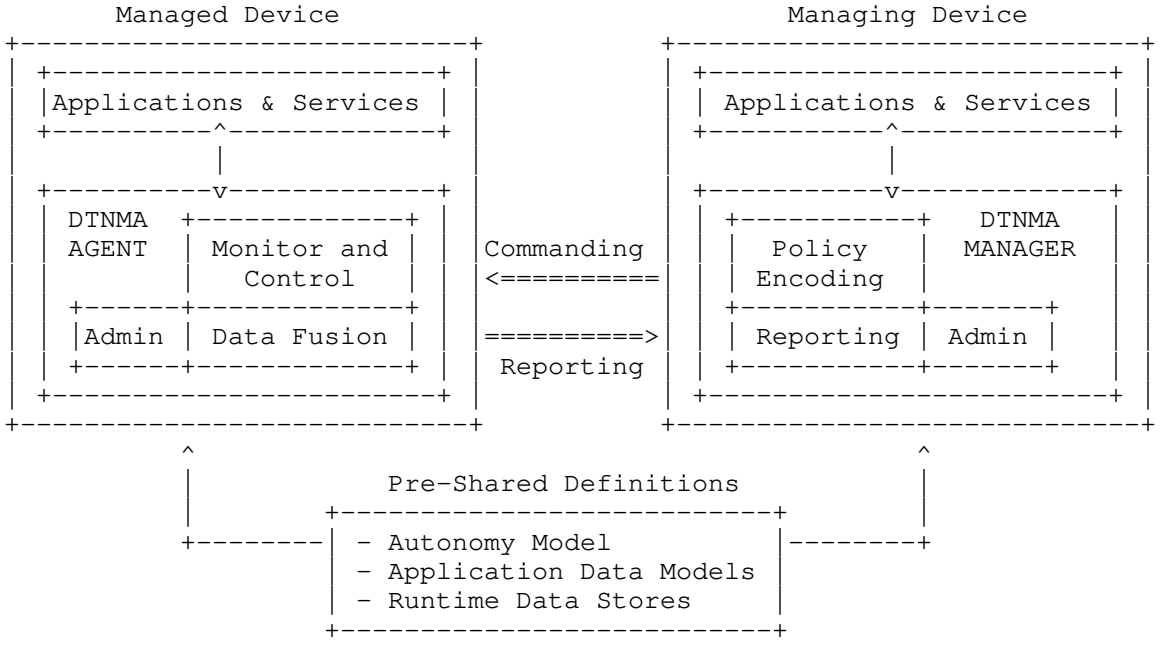


Figure 1

This model preserves the familiar concept of "managers" resident on managing devices and "agents" resident on managed devices. However, the DTNMA model is unique in how the DM and DA operate. The DM is used to pre-configure DAs in the network with management policies. it is expected that the DAs, themselves, perform monitoring and control functions on their own. In this way, a properly configured DA may operate without a reliable connection back to a DM.

7.3. Functional Elements

The reference model illustrated in Figure 1 implies the existence of certain logical components whose roles and responsibilities are discussed in this section.

7.3.1. Managed Applications and Services

By definition, managed applications and services reside on a managed device. These software entities can be controlled through some interface by the DA and their state can be sampled as part of periodic monitoring. It is presumed that the DA on the managed device has the proper data model, control interface, and permissions to alter the configuration and behavior of these software applications.

7.3.2. DTNMA Agent (DA)

A DA resides on a managed device. As is the case with other network management approaches, this agent is responsible for the monitoring and control of the applications local to that device. Unlike other network management approaches, the agent accomplishes this task without a regular connection to a DTNMA Manager.

The DA performs three major functions on a managed device: the monitoring and control of local applications, production of data analytics, and the administrative control of the agent itself.

7.3.2.1. Monitoring and Control

DAs monitor the status of applications running on their managed device and selectively control those applications as a function of that monitoring. The following components are used to perform monitoring and control on an agent.

Rules Database:
: Each DA maintains a database of policy expressions that form rules of behavior of the managed device. Within this database, each rule of behavior is a tuple of a stimulus and a response. Within the DTNMA, these rules are the embodiment of policy expressions received from DMs and evaluated at regular intervals by the autonomy engine. The rules database is the collection of active rules known to the DA.

Autonomy Engine:
: The DA autonomy engine monitors the state of the managed device looking for pre-defined stimuli and, when encountered, issuing a pre-defined response. To the extent that this function is driven by the rules database, this engine acts as a policy execution engine. This engine may also be directly configured by managers during periods of connectivity for actions separate from those in the rules database (such as enabling or disabling sets of rules). Once configured, the engine may function without other access to any managing device. This engine may also reconfigure itself as a function of policy.

Application Control Interfaces:
: DAs support control interfaces for all managed applications. Control interfaces are used to alter the configuration and behavior of an application. These interfaces may be custom for each application, or as provided through a common framework such as provided by an operating system.

7.3.2.2. Data Fusion

DAs generate new data elements as a function of the current state of the managed device and its applications. These new data products may take the form of individual data values, or new collections of data used for reporting. The logical components responsible for these behaviors are as follows.

Application Data Interfaces:
: DAs support mechanisms by which important state is retrieved from various applications resident on the managed device. These data interfaces may be custom for each application, or as provided through a common framework such as provided by an operating system.

Data Value Generators:
: DAs may support the generation of new data values as a function of other values collected from the managed device. These data generators may be configured with descriptions of data values and the data values they generate may be included in the overall monitoring and reporting associated with the managed device.

Report Generators:
: DAs may, as appropriate, generate collections of data values and provide them to whatever local mechanism takes responsibility for their eventual transmission (or expiration and removal). Reports can be generated as a matter of policy or in response to the handling of critical events (such as errors), or other logging needs. The generation of a report is independent of whether there exists any connectivity between a DA and a DM.

7.3.2.3. Administration

DAs perform a variety of administrative services in support of their configuration. The significant such administrative services are as follows.

Manager Mapping:
: The DTNMA allows for a many-to-many relationship amongst DTNMA Agents and Managers. A single DM may configure multiple DAs, and a single DA may be configured by multiple DMs. Multiple managers may exist in a network for at least two reasons. First, different managers may exist to control different applications on a device. Second, multiple managers increase the likelihood of an agent encountering a manager when operating in a sparse or challenged environment.

While the need for multiple managers is required for operating in a dynamically partitioned network, this situation allows for the possibility of conflicting information from different managers. Implementations of the DTNMA should consider conflict resolution mechanisms. Such mechanisms might include analyzing managed content, time, agent location, or other relevant information to select one manager input over other manager inputs.

Data Verifiers:
: DAs might handle large amounts of data produced by various sources, to include data from local managed applications, remote managers, and self-calculated values. DAs should ensure, when possible, that externally generated data values have the proper syntax and semantic constraints (e.g., data type and ranges) and any required authorization.

Access Controllers:
: DAs support authorized access to the management of individual applications, to include the administrative management of the agent itself. This means that a manager may only set policy on the agent pursuant to verifying that the manager is authorized to do so.

7.3.3. Managing Applications and Services

Managing applications and services reside on a managing device and serve as the both the source of DA policy statements and the target of DA reporting. They may operate with or without an operator in the loop.

Unlike management applications in unchallenged networks, these applications cannot exert closed-loop control over any managed device application. Instead, they exercise open-loop control by producing policies that can be configured and enforced on managed devices by DAs.

> NOTE: Closed-loop control in this context refers to the practice of waiting for a response from a managed device prior to issuing new commands to that device. These "loops" may be closed quickly (in milliseconds) or over much longer periods (hours, days, years). The alternative to closed-loop control is open-loop control, where the issuance of new commands is not dependent on receiving responses to previous commands. Additionally, there might not be a 1-1 mapping between commands and responses. A DA may, for example, produce a single response that captures the end state from applying multiple commands.

7.3.4. DTNMA Manager (DM)

A DM resides on a managing device. This manager provides an interface between various managing applications and services and the DAs that enforce their policies. In providing this interface, DMs translate between whatever native interface exists to various managing applications and the autonomy models used to encode management policy.

The DM performs three major functions on a managing device: policy encoding, reporting, and administration.

7.3.4.1. Policy Encoding

DMs translate policy directives from managing applications and services into standardized policy expressions that can be recognized by DAs. The following logical components are used to perform this policy encoding.

Application Control Interfaces:
: DMs support control interfaces for managing applications. These control interfaces are used to receive desired policy statements from applications. These interfaces may be custom for each application, or provided through a common framework, protocol, or operating system.

Policy Encoders:
: DAs implement a standardized autonomy model comprising standardized data elements. This allows the open-loop control structures provided by managing applications to be represented in a common language. Policy encoders perform this encoding function.

Policy Aggregators:
: DMs collect multiple encoded policies into messages that can be sent to DAs over the network. This implies the proper addressing of agents and the creation of messages that support store-and-forward operations. It is recommended that control messages be packaged using BP bundles when there may be intermittent connectivity between DMs and DAs.

7.3.4.2. Reporting

DMs receive reports on the status of managed devices during periods of connectivity with the DAs on those devices. The following logical components are needed to implement reporting capabilities on a DM.

Report Collectors:
: DMs receive reports from DAs in an asynchronous manner. This means that reports may be received out of chronological order and in ways that are difficult or impossible to associate with a specific policy from a managing application. DMs collect these reports and extract their data in support of subsequent data analytics.

Data Analyzers:
: DMs review sets of data reports from DAs with the purpose of extracting relevant data to communicate with managing applications. This may include simple data extraction or may include more complex processing such as data conversion, data fusion, and appropriate data analytics.

Application Data Interfaces:
: DMs support mechanisms by which data retrieved from agent may be provided back to managing devices. These interfaces may be custom for each application, or as provided through a common framework, protocol, or operating system.

7.3.4.3. Administration

Managers in the DTNMA perform a variety of administrative services in support of their proper configuration and operation. This includes the following logical components.

Agent Mappings:
: The DTNMA allows DMs to communicate with multiple DAs. However, not every agent in a network is expected to support the same set of Application Data Models or otherwise have the same set of managed applications running. For this reason, DMs determine individual DA capabilities to ensure that only appropriate Controls are sent to a DA.

Data Verifiers:
: DMs handle large amounts of data produced by various sources, to include data from managing applications and DAs. DMs should ensure, when possible, that data values received from DAs over a network have the proper syntax and semantic constraints (e.g., data type and ranges) and any required authorization.

Access Controllers:
: DMs should only send Controls to agents when the manager is configured with appropriate access to both the agent and the applications being managed.

7.3.5. Pre-Shared Definitions

A consequence of operating in a challenged environment is the potential inability to negotiate information in real-time. For this reason, the DTNMA requires that managed and managing devices operate using pre-shared definitions rather than relying on data definition negotiation.

The three types of pre-shared definitions in the DTNMA are the DA autonomy model, managed application data models, and any runtime data shared by managers and agents.

Autonomy Model:
: A DTNMA autonomy model represents the data elements and associated autonomy structures that define the behavior of the agent autonomy engine. A standardized autonomy model allows for individual implementations of DAs, and DMs to interoperate. A standardized model also provides guidance to the design and implementation of both managed and managing applications.

Application Data Models:
: As with other network management architectures, the DTNMA pre-supposes that managed applications (and services) define their own data models. These data models include the data produced by, and Controls implemented by, the application. These models are expected to be static for individual applications and standardized for applications implementing standard protocols.

Runtime Data Stores:
: Runtime data stores, by definition, include data that is defined at runtime. As such, the data is not pre-shared prior to the deployment of DMs and DAs. Pre-sharing in this context means that DMs and DAs are able to define and synchronize data elements prior to their operational use in the system. This synchronization happens during periods of connectivity between DMs and DAs.

8. Desired Services

This section provides a description of the services provided by DTNMA components on both managing and managed devices. These service descriptions differ from other management descriptions because of the unique characteristics of the DTNMA operating environment.

> Predicate autonomy, asynchronous data transport, and intermittent connectivity require new techniques for device management. Many of the services discussed in this section attempt to provide continuous operation of a managed device through periods of no connectivity.

8.1. Local Monitoring and Control

DTNMA monitoring is associated with the agent autonomy engine. The term monitoring implies regular access to information such that state changes may be acted upon within some response time period. Within the DTNMA, connections between a managed and managing device are unable to provide such a connection and, thus, monitoring functions are performed on the managed device.

Predicate autonomy on a managed device should collect state associated with the device at regular intervals and evaluate that collected state for any changes that require a preventative or corrective action. Similarly, this monitoring may cause the device to generate one or more reports destined to the managing device.

Similar to monitoring, DTNMA control results in actions by the agent to change the state or behavior of the managed device. All control in the DTNMA is local control. In cases where there exists a timely connection to a manager, received Controls are still run through the autonomy engine. In this case, the stimulus is the direct receipt of the Control and the response is to immediately run the Control. In this way, there is never a dependency on a session or other stateful exchange with any remote entity.

8.2. Local Data Fusion

DTNMA Fusion services produce new data products from existing state on the managed device. These fusion products can be anything from simple summations of sampled counters to complex calculations of behavior over time.

Fusion is an important service in the DTNMA because fusion products are part of the overall state of a managed device. Complete knowledge of this overall state is important for the management of the device, particularly in a stimulus-response system whose stimuli are evaluated against this state. In particular, the predicates of rules on a DA may refer to fused data.

In-situ data fusion is an important function as it allows for the construction of intermediate summary data, the reduction of stored and transmitted raw data, possibly fewer predicates in rule definitions, and otherwise insulates the data source from conclusions drawn from that data.

While some fusion is performed in any management system, the DTNMA requires fusion to occur on the managed device itself. If the network is partitioned such that no connection to a managing device is available, fusion happens locally. Similarly, connections to a managing device might not remain active long enough for round-trip data exchange or may not have the bandwidth to send all sampled data.

> NOTE: While data fusion is an important function within the DTNMA, it is expected that the storage and transmission of raw (or pre-fused) data remains a capability of the system. In particular, raw data can be useful for debugging managed devices, understanding complex interactions and underlying conditions, and tuning for better performance and/or better outcomes.

8.3. Remote Configuration

DTNMA configuration services update the local configuration of a managed device with the intent to impact the behavior and capabilities of that device. The change of device configurations is a common service provided by many network management systems. The DTNMA has a unique approach to configuration for the following reasons.

The DTNMA configuration service is unique in that the selection of managed device configurations occurs, itself, as a function of the state of the device. This implies that management proxies on the device store multiple configuration functions that can be applied as needed without consultation from a managing device.

> This approach differs from the management concept of selecting from multiple datastores in that DTNMA configuration functions can target individual data elements and can calculate new values from local device state.

When detecting stimuli, the agent autonomy engine supports a mechanism for evaluating whether application monitoring data or runtime data values are recent enough to indicate a change of state. In cases where data has not been updated recently, it may be considered stale and not used to reliably indicate that some stimulus has occurred.

8.4. Remote Reporting

DTNMA reporting services collect information known to the managed device and prepare it for eventual transmission to one or more managing devices. The contents of these reports, and the frequency at which they are generated, occurs as a function of the state of the managed device, independent of the managing device.

Once generated, it is expected that reports might be queued pending a connection back to a managing device. Therefore, reports need to be differentiable as a function of the time they were generated.

> NOTE: When reports are queued pending transmission, the overall storage capacity at the queuing device needs to be considered. There may be cases where queued reports can be considered expired either because they have been queued for too long, or because they have been replaced by a newer report. When a report is considered expired, it may be considered for removal and, thus, never transmitted. This consideration is expected to be part of the implementation of the queuing device and not the responsibility of the reporting function within the DTNMA.

When reports are sent to a managing device over a challenged network, they may arrive out of order due to taking different paths through the network or being delayed due to retransmissions. A managing device should not infer meaning from the order in which reports are received.

Reports may or may not be associated with a specific Control. Some reports may be annotated with the Control that caused the report to be generated. Sometimes, a single report will represent the end state of applying multiple Controls.

8.5. Authorization

Both local and remote services provided by the DTNMA affect the behavior of multiple applications on a managed device and may interface with multiple managing devices.

Authorization services enforce the potentially complex mapping of other DTNMA services amongst managed and managing devices in the network. For example, fine-grained access control can determine which managing devices receive which reports, and what Controls can be used to alter which managed applications.

This is particularly beneficial in networks that either deal with multiple administrative entities or overlay networks that cross administrative boundaries. Allowlists, blocklists, key-based infrastructures, or other schemes may be used for this purpose.

9. Logical Autonomy Model

An important characteristic of the DTNMA is the shift in the role of a managing device. In the DTNMA, managers configure the autonomy engines on agents, and it is the agents that provide local device management. One way to describe the behavior of the agent autonomy engine is to describe the characteristics of the autonomy model it implements.

This section describes a logical autonomy model in terms of the abstract data elements that would comprise the model. Defining abstract data elements allows for an unambiguous discussion of the behavior of an autonomy model without mandating a particular design, encoding, or transport associated with that model.

9.1. Overview

A managing autonomy capability on a potentially disconnected device needs to behave in both an expressive and deterministic way. Expressivity allows for the model to be configured for a wide range of future situations. Determinism allows for the forensic reconstruction of device behavior as part of debugging or recovery efforts. It also is necessary to ensure predictable behavior.

> NOTE: The use of predicate logic and a stimulus-response system does not conflict with the use of higher-level autonomous function or the incorporation of machine learning. Specifically, the DTNMA deterministic autonomy model can coexist with other autonomous functions managing applications and network services.
>
> An example of such co-existence is the use of the DTNMA model to ensure a device stays within safe operating parameters while a less deterministic machine learning model directs smaller behaviors for the device.

The DTNMA autonomy model is a rule-based model in which individual rules associate a pre-identified stimulus with a pre-configured response to that stimulus.

Stimuli are identified using one or more predicate logic expressions that examine aspects of the state of the managed device. Responses are implemented by running one or more procedures on the managed device.

In its simplest form, a stimulus is a single predicate expression of a condition that examines some aspect of the state of the managed device. When the condition is met, a predetermined response is applied. This behavior can be captured using the construct:

```
IF <condition 1> THEN <response 1>;
```

In more complex forms, a stimulus may include both a common condition shared by multiple rules and a specific condition for each individual rule. If the common condition is not met, the evaluation of the specific condition of each rule sharing the common condition can be skipped. In this way, the total number of predicate evaluations can be reduced. This behavior can be captured using the construct:

```
IF <common condition> THEN
  IF <specific condition 1> THEN <response 1>
  IF <specific condition 2> THEN <response 2>
  IF <specific condition 3> THEN <response 3>
```

> NOTE: The DTNMA model remains a stimulus-response system, regardless of whether a common condition is part of the stimulus. However, it is recommended that implementations incorporate a common condition because of the efficiency provided by such a bulk evaluation.
>
> NOTE: One use of a stimulus "common condition" is to associated the condition with an on-board event such as the expiring of a timer or the changing of a monitored value.
>
> NOTE: The DTNMA does not prescribe when to evaluate rule stimuli. Implementations may choose to evaluate rule stimuli at periodic intervals (such as 1Hz or 100Hz). When stimuli include on-board events, implementations may choose to perform an immediate evaluation at the time of the event rather than waiting for a periodic evaluation.

DTNMA Autonomy Model

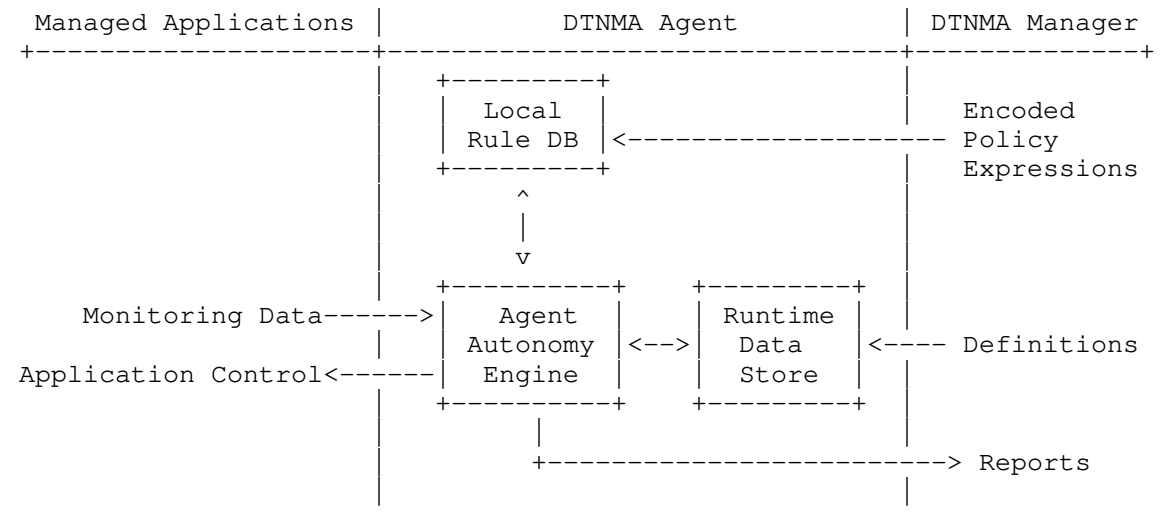


Figure 2

The flow of data into and out of the agent autonomy engine is illustrated in Figure 2. In this model, the autonomy engine stores the combination of stimulus conditions and associated responses as a set of "rules" in a rules database. This database is updated through the execution of the autonomy engine and as configured from policy statements received by managers.

Stimuli are detected by examining the state of applications as reported through application monitoring interfaces and through any locally-derived data. Local data is calculated in accordance with definitions also provided by managers as part of the runtime data store.

Responses to stimuli are run as updated to the rules database, updated to the runtime data store, Controls sent to applications, and the generation of reports.

9.2. Model Characteristics

There are several practical challenges to the implementation of a distributed rule-based system. Large numbers of rules may be difficult to understand, deconflict, and debug. Rules whose conditions are given by fused or other dynamic data may require data logging and reporting for deterministic offline analysis. Rule differences across managed devices may lead to oscillating effects. This section identifies those characteristics of an autonomy model that might help implementations mitigate some of these challenges.

There are a number of ways to represent data values, and many data modeling languages exist for this purpose. When considering how to model data in the context of the DTNMA autonomy model there are some modeling features that should be present to enable functionality. There are also some modeling features that should be prevented to avoid ambiguity.

Traditional network management approaches favor flexibility in their data models. The DTNMA stresses deterministic behavior that supports forensic analysis of agent activities "after the fact". As such, the following statements should be true of all data representations relating to DTNMA autonomy.

Strong Typing: The predicates and expressions that comprise the autonomy services in the DTNMA should require strict data typing. This avoids errors associated with implicit data conversions and helps detect misconfiguration.

Acyclic Dependency: Many dependencies exist in an autonomy model, particularly when combining individual expressions or results to create complex behaviors. Implementations that conform to the DTNMA need to prevent circular dependencies.

Fresh Data: Autonomy models operating on data values presume that their data inputs represent the actionable state of the managed device. If a data value has failed to be refreshed within a time period, autonomy might incorrectly infer an operational state. Regardless of whether a data value has changed, DTNMA implementations should provide some indicator of whether the data value is "fresh" meaning that it still represents the current state of the device.

Pervasive Parameterization: Where possible, autonomy model objects should support parameterization to allow for flexibility in the specification. Parameterization allows for the definition of fewer unique model objects and also can support the substitution of local device state when exercising device control or data reporting.

Configurable Cardinality: The number of data values that can be supported in a given implementation is finite. For devices operating in challenged environments, the number of supported objects may be far fewer than that which can be supported by devices in well-resourced environments. DTNMA implementations should define limits to the number of supported objects that can be active in a system at one time, as a function of the resources available to the implementation.

Control-Based Updates: The agent autonomy engine changes the state of the managed device by running Controls on the device. This is different from other approaches where the behavior of a managed device is updated only by updated configuration values, such as in a table or datastore. Altering behavior via one or more Controls allows checking all pre-conditions before making changes as well as providing more granularity in the way in which the device is updated. Where necessary, Controls can be defined to perform bulk updates of configuration data so as not to lose that update modality. One important update pre-condition is that the system is not performing an action that would prevent the update (such as currently applying a competing update).

9.3. Data Value Representation

The expressive representation of simple data values is fundamental to the successful construction and evaluation of predicates in the DTNMA autonomy model. When defining such values, there are useful distinctions regarding how values are identified and whether values are generated internal or external to the autonomy model.

A DTNMA data value should combine a base type (e.g., integer, real, string) representation with relevant semantic information. Base types are used for proper storage and encoding. Semantic information allows for additional typing, constraint definitions, and mnemonic naming. This expanded definition of data value allows for better predicate construction and evaluation, early type checking, and other uses.

Data values may further be annotated based on whether their value is the result of a DA calculation or the result of some external process on the managed device. For example, operators may with to know which values can be updated by actions on the DA versus which values (such as sensor readings) cannot be reliably changed because they are calculated external to the DA.

9.4. Data Reporting

The DTNMA autonomy model should, as required, report on the state of its managed device (to include the state of the model itself). This reporting should be done as a function of the changing state of the managed device, independent of the connection to any managing device. Queuing reports allows for later forensic analysis of device behavior, which is a desirable property of DTNMA management.

DTNMA data reporting consists of the production of some data report instance conforming to a data report schema. The use of schemas allows a report instance to identify the schema to which it conforms in lieu of carry that structure in the instance itself. This approach can significantly reduce the size of generated reports.

> NOTE: The DTNMA data reporting concept is intentionally distinct from the concept of exchanging data stores across a network. It is envisioned that a DA might generate a data report instance of a data report schema at regular intervals or in response to local events. In this model, many report schemas may be defined to capture unique, relevant combinations of known data values rather than sending bulk data stores off-platform for analysis.
>
> NOTE: It is not required that data report schemas be tabular in nature. Individual implementations might define tabular schemas for table-like data and other report schemas for more heterogeneous reporting.

9.5. Command Execution

The agent autonomy engine requires that managed devices issue commands on themselves as if they were otherwise being controlled by a managing device. The DTNMA implements commanding through the use of Controls and macros.

Controls represent parameterized, predefined procedures run by the DA either as directed by the DM or as part of a rule response from the DA autonomy engine. Macros represent ordered sequences of Controls.

Controls are conceptually similar to RPCs in that they represent parameterized functions run on the managed device. However, they are conceptually dissimilar from RPCs in that they do not have a concept of a return code because they operate over an asynchronous transport. The concept of return code in an RPC implies a synchronous relationship between the caller of the procedure and the procedure being called, which might not be possible within the DTNMA.

The success or failure of a Control may be handled locally by the agent autonomy engine. Local error handling is particularly important in this architecture given the potential for long periods of disconnectivity between a DA and a DM. The failure of one or more Controls on a DA represent part of the state of the DA and, therefore, able to trigger rules as part of the Agent autonomy engine.

The impact of a Control is externally observable via the generation and eventual examination of data reports produced by the managed device.

The failure of certain Controls might leave a managed device in an undesired state. Therefore, it is important that there be consideration for Control-specific recovery mechanisms (such as a rollback or safing mechanism). When a Control that is part of a macro (such as in an autonomy response) fails, there may be a need to implement a safe state for the managed device based on the nature of the failure.

> NOTE: The use of the term Control in the DTNMA is derived in part from the concept of Command and Control (C2) where control implies the operational instructions undertaken to implement (or maintain) a commanded objective. The DA autonomy engine controls a managed device to allow it to fulfill some purpose as commanded by a (possibly disconnected) managing device.
>
> For example, attempting to maintain a safe internal thermal environment for a spacecraft is considered "thermal control" (not "thermal commanding") even though thermal control involves sending commands to heaters, louvers, radiators, and other temperature-affecting components.
>
> Even when Controls are received from a managing device with the intent to be run immediately, the control-vs-command distinction still applies. The Control being run on the managed device is in service of the command received from the managing device to immediately change the local state of the device.

9.6. Predicate Autonomy Rules

As discussed in Section 9.1, the DTNMA rule-based stimulus-response system associates stimulus detection with a predetermined response. Rules may be categorized based on whether their stimuli include generic statements of managed device state or whether they are optimized to only consider the passage of time on the device.

State-based rules are those whose stimulus is based on the evaluated state of the managed device. Time-based rules are a unique subset of state-based rules whose stimulus is given only by a time-based event. Implementations might create different structures and evaluation mechanisms for these two different types of rules to achieve more efficient processing on a platform.

10. Use Cases

Using the autonomy model defined in Section 9, this section describes flows through sample configurations conforming to the DTNMA. These use cases illustrate remote configuration, local monitoring and control, multiple manager support, and data fusion.

10.1. Notation

The use cases presented in this section are documented with a shorthand notation to describe the types of data sent between managers and agents. This notation, outlined in Table 1, leverages the definitions of autonomy model components defined in Section 9.

Term	Definition	Example
EDD#	Externally Defined Data - a data value defined external to the DA.	EDD1, EDD2
V#	Variable - a data value defined internal to the DA.	V1 = EDD1 + 7
EXPR	Predicate expression - used to define a rule stimulus.	V1 > 5
ID	DTNMA Object Identifier.	V1, EDD2
ACL#	Enumerated Access Control List.	ACL1
DEF(ACL,ID,EXPR)	Define ID from expression. Allow managers in ACL to see this ID.	DEF(ACL1, V1, EDD1 + EDD2)
PROD(P,ID)	Produce ID according to predicate P. P may be a time period (1s) or an expression (EDD1 > 10).	PROD(1s, EDD1)
RPT(ID)	A report instance containing data named ID.	RPT(EDD1)

Table 1: Terminology

These notations do not imply any implementation approach. They only provide a succinct syntax for expressing the data flows in the use case diagrams in the remainder of this section.

10.2. Serialized Management

This nominal configuration shows a single DM interacting with multiple DAs. The control flows for this scenario are outlined in Figure 3.

Serialized Management Control Flow

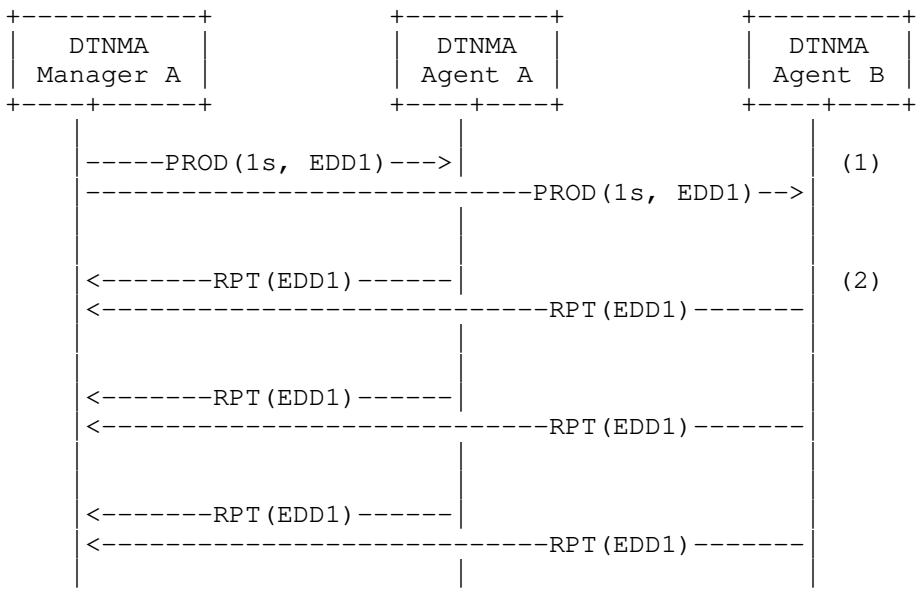


Figure 3

In a serialized management scenario, a single DM interacts with multiple DAs.

In this figure, the DTNMA Manager A sends a policy to DTNMA Agents A and B to report the value of an EDD (EDD1) every second in (step 1). Each DA receives this policy and configures their respective autonomy engines for this production. Thereafter, (step 2) each DA produces a report containing data element EDD1 and sends those reports back to the DM.

This behavior continues without any additional communications from the DM and without requiring a connection between the DA and DM.

10.3. Intermittent Connectivity

Building from the nominal configuration in Section 10.2, this scenario shows a challenged network in which connectivity between DTNMA Agent B and the DM is temporarily lost. Control flows for this case are outlined in Figure 4.

Challenged Management Control Flow

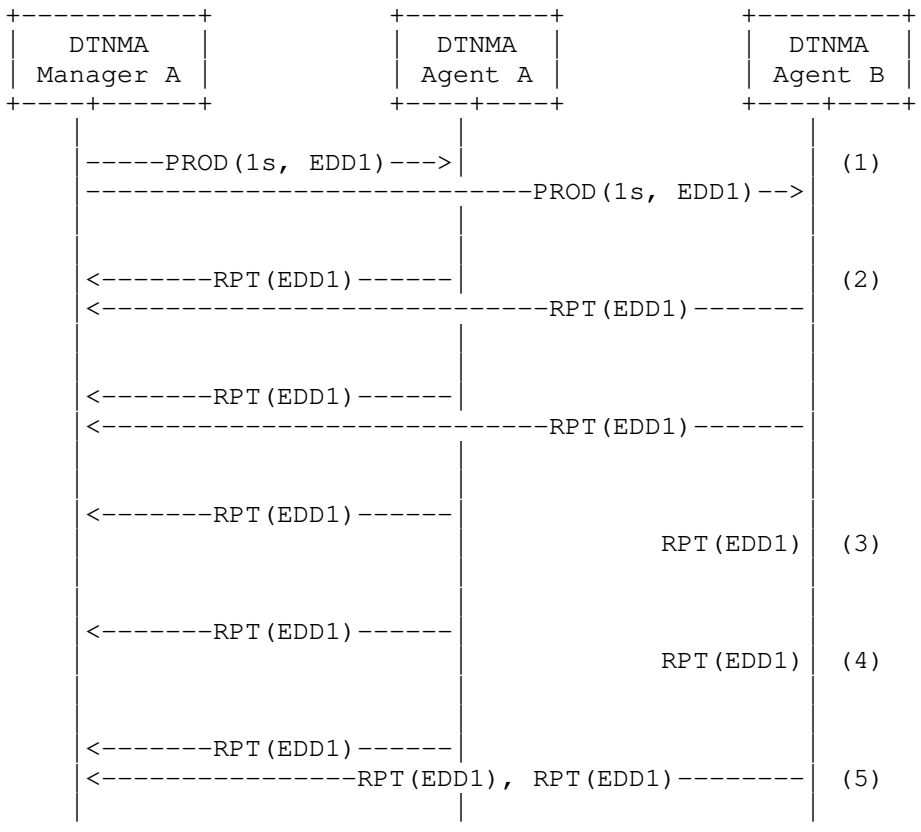


Figure 4

In a challenged network, DAs store reports pending a transmit opportunity.

In this figure, DTNMA Manager A sends a policy to DTNMA Agents A and B to produce an EDD (EDD1) every second in (step 1). Each DA receives this policy and configures their respective autonomy engines for this production. Produced reports are transmitted when there is connectivity between the DA and DM (step 2).

At some point, DTNMA Agent B loses the ability to transmit in the network (steps 3 and 4). During this time period, DA B continues to produce reports, but they are queued for transmission. This queuing might be done by the DA itself or by a supporting transport such as BP. Eventually (and before the next scheduled production of EDD1), DTNMA Agent B is able to transmit in the network again (step 5) and all queued reports are sent at that time. DTNMA Agent A maintains connectivity with the DM during steps 3-5, and continues to send reports as they are generated.

10.4. Open-Loop Reporting

This scenario illustrates the DTNMA open-loop control paradigm, where DAs manage themselves in accordance with policies provided by DMs, and provide reports to DMs based on these policies.

The control flow shown in Figure 5, includes an example of data fusion, where multiple policies configured by a DM result in a single report from a DA.

Consolidated Management Control Flow

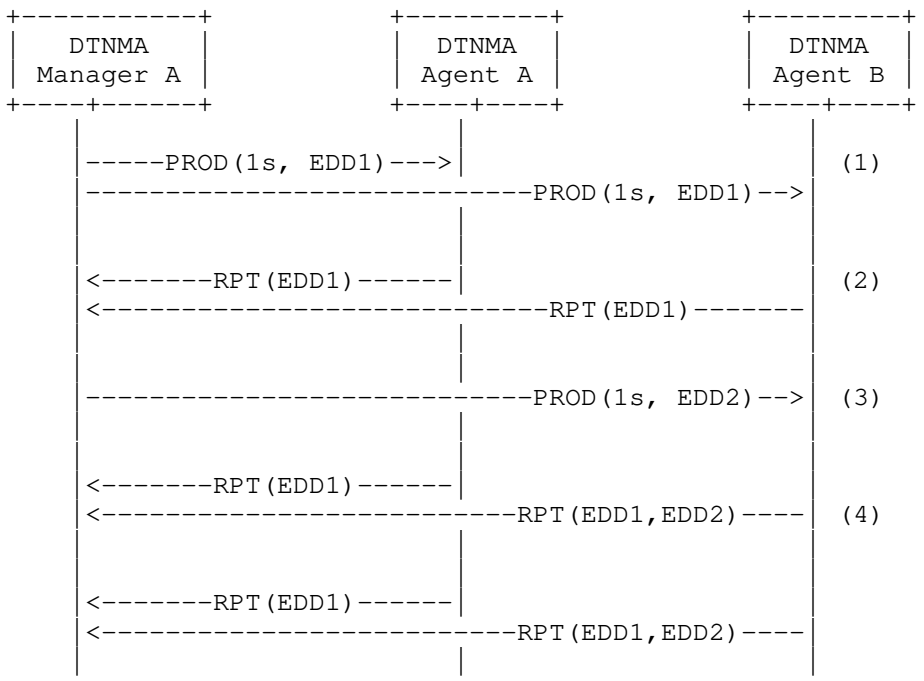


Figure 5

A many-to-one mapping between management policy and device state reporting is supported by the DTNMA.

In this figure, DTNMA Manager A sends a policy statement in the form of a rule to DTNMA Agents A and B, which instructs the DAs to produce a report with EDD1 every second (step 1). Each DA receives this policy, which is stored in its respective Rule Database, and configures its Autonomy Engine. Reports are transmitted by each DA when produced (step 2).

At a later time, DTNMA Manager A sends an additional policy to DTNMA Agent B, requesting the production of a report for EDD2 every second (step 3). This policy is added to DTNMA Agent B's Rule Database.

Following this policy update, DTNMA Agent A will continue to produce EDD1 and DTNMA Agent B will produce both EDD1 and EDD2 (step 4). However, DTNMA Agent B may provide these values to the DM in a single report rather than as 2 independent reports. In this way, there is no direct mapping between the single consolidated report sent by DTNMA Agent B (step 4) and the two different policies sent to DTNMA Agent B that caused that report to be generated (steps 1 and 3).

10.5. Multiple Administrative Domains

The managed applications on a DA may be controlled by different administrative entities in a network. The DTNMA allows DAs to communicate with multiple DMs in the network, such as in cases where there is one DM per administrative domain.

Whenever a DM sends a policy expression to a DA, that policy expression may be associated with authorization information. One method of representing this is an ACL.

> The use of an ACL in this use case does not imply the DTNMA requires ACLs to annotate policy expressions. Further, the inclusion of ACLs in the policy expressions themselves is for representation purposes only, as ACLs are internal to DAs and not supplied explicitly in messaging. ACLs and their representation in this context are for example purposes only.

The ability of one DM to access the results of policy expressions configured by some other DM will be limited to the authorization annotations of those policy expressions.

An example of multi-manager authorization is illustrated in Figure 6.

Multiplexed Management Control Flow

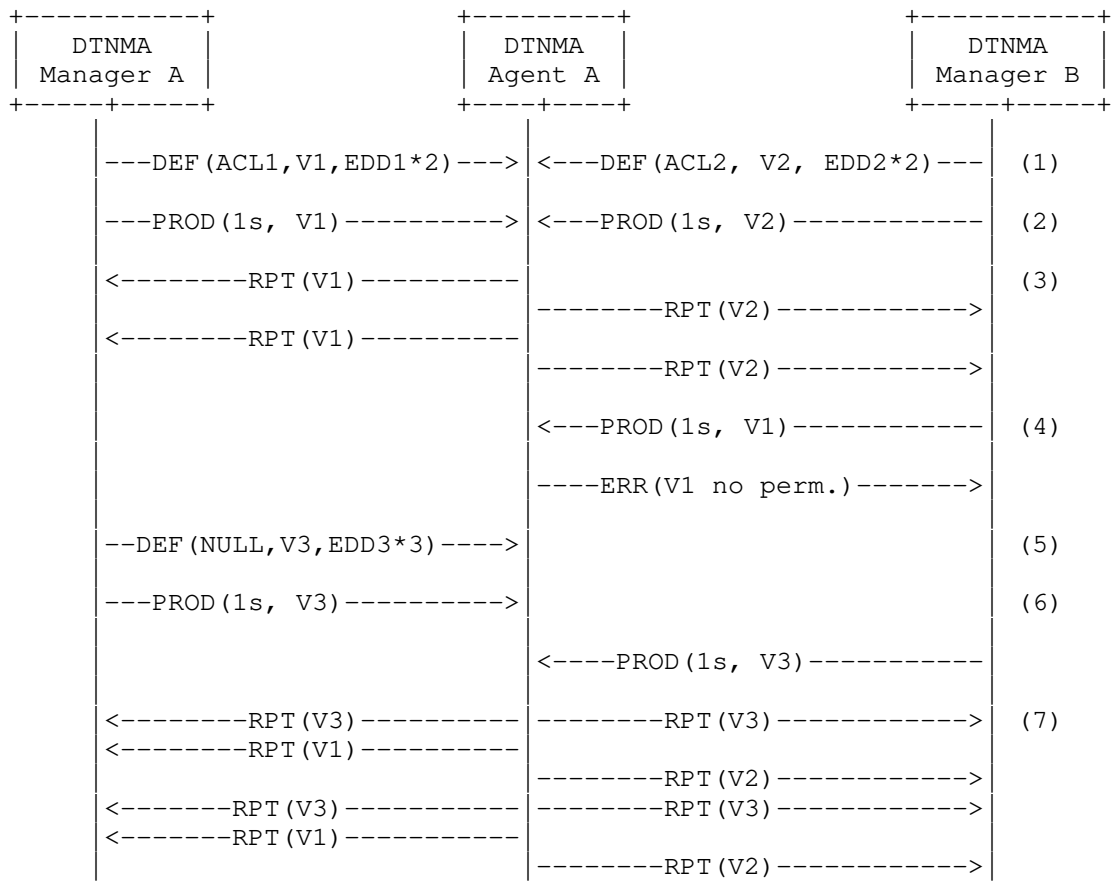


Figure 6

Multiple DMs may interface with a single DA, particularly in complex networks.

In this figure, both DTNMA Managers A and B send policies to DTNMA Agent A (step 1). DM A defines a variable (V1) whose value is given by the mathematical expression (EDD1 * 2) and is associated with an ACL (ACL1) that restricts access to V1 to DM A only. Similarly, DM B defines a variable (V2) whose value is given by the mathematical expression (EDD2 * 2) and associated with an ACL (ACL2) that restricts access to V2 to DM B only.

Both DTNMA Managers A and B also send policies to DTNMA Agent A to report on the values of their variables at 1 second intervals (step 2). Since DM A can access V1 and DM B can access V2, there is no authorization issue with these policies and they are both accepted by the autonomy engine on Agent A. Agent A produces reports as expected, sending them to their respective managers (step 3).

Later (step 4) DM B attempts to configure DA A to also report to it the value of V1. Since DM B does not have authorization to view this variable, DA A does not include this in the configuration of its autonomy engine and, instead, some indication of permission error is included in any regular reporting back to DM B.

DM A also sends a policy to Agent A (step 5) that defines a variable (V3) whose value is given by the mathematical expression (EDD3 * 3) and is not associated with an ACL, indicating that any DM can access V3. In this instance, both DM A and DM B can then send policies to DA A to report the value of V3 (step 6). Since there is no authorization restriction on V3, these policies are accepted by the autonomy engine on Agent A and reports are sent to both DM A and B over time (step 7).

10.6. Cascading Management

There are times where a single network device may serve as both a DM for other DAs in the network and, itself, as a device managed by someone else. This may be the case on nodes serving as gateways or proxies. The DTNMA accommodates this case by allowing a single device to run both a DA and DM.

An example of this configuration is illustrated in Figure 7.

Data Fusion Control Flow

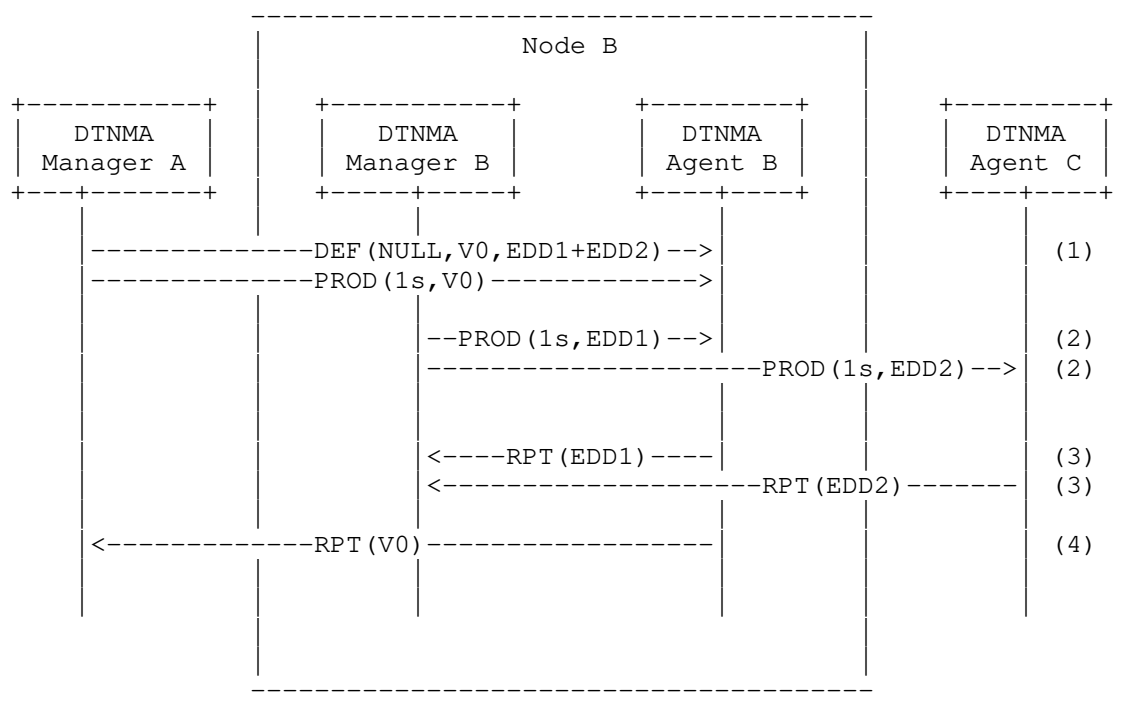


Figure 7

A device can operate as both a DTNMA Manager and an Agent.

In this example, we presume that DA B is able to sample a given EDD (EDD1) and that DA C is able to sample a different EDD (EDD2). Node B houses DM B (which controls DA C) and DA B (which is controlled by DM A). DM A must periodically receive some new value that is calculated as a function of both EDD1 and EDD2.

First, DM A sends a policy to DA B to define a variable (V0) whose value is given by the mathematical expression (EDD1 + EDD2) without a restricting ACL. Further, DM A sends a policy to DA B to report on the value of V0 every second (step 1).

DA B can require the ability to monitor both EDD1 and EDD2. However, the only way to receive EDD2 values is to have them reported back to Node B by DA C and included in the Node B runtime data stores. Therefore, DM B sends a policy to DA C to report on the value of EDD2 (step 2).

DA C receives the policy in its autonomy engine and produces reports on the value of EDD2 every second (step 3).

DA B may locally sample EDD1 and EDD2 and uses that to compute values of V0 and report on those values at regular intervals to DM A (step 4).

While a trivial example, the mechanism of associating fusion with the Agent function rather than the Manager function scales with fusion complexity. Within the DTNMA, DAs and DMs are not required to be separate software implementations. There may be a single software application running on Node B implementing both DM B and DA B roles.

11. IANA Considerations

This document requires no IANA actions.

12. Security Considerations

Security within a DTNMA exists in at least two layers: security in the data model and security in the messaging and encoding of the data model.

Data model security refers to the validity and accessibility of data elements. For example, a data element might be available to certain DAs or DMs in a system, whereas the same data element may be hidden from other DAs or DMs. Both verification and authorization mechanisms at DAs and DMs are important to achieve this type of security.

> NOTE: One way to provide finer-grained application security is through the use of Access Control Lists (ACLs) that would be defined as part of the configuration of DAs and DMs. It is expected that many common data model tools provide mechanisms for the definition of ACLs and best practices for their operational use.

The exchange of information between and amongst DAs and DMs in the DTNMA is expected to be accomplished through some secured messaging transport.

13. Informative References

[ASN.1] International Organization for Standardization, "Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)", International Standard 8824, December 1987.

[DART] Tropf, B. T., Haque, M., Behrooz, N., and C. Krupiarz, "The DART Autonomy System", 2023, <https://ieeexplore.ieee.org/abstract/document/10207457>.

[gNMI] OpenConfig, "gRPC Network Management Interface (gNMI)", May 2023, <https://www.openconfig.net/docs/gnmi/gnmi-specification/>.

[gRPC] gRPC Authors, "gRPC Documentation", 2024, <https://grpc.io/docs/>.

[I-D.ietf-core-comi] Veillette, M., Van der Stok, P., Pelov, A., Bierman, A., and C. Bormann, "CoAP Management Interface (CORECONF)", Work in Progress, Internet-Draft, draft-ietf-core-comi-16, 4 September 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-comi-16>.

[I-D.ietf-core-sid] Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-24, 22 December 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-sid-24>.

[I-D.rfernando-protocol-buffers] Stuart, S. and R. Fernando, "Encoding rules and MIME type for Protocol Buffers", Work in Progress, Internet-Draft, draft-rfernando-protocol-buffers-00, 11 October 2012, <https://datatracker.ietf.org/doc/html/draft-rfernando-protocol-buffers-00>.

[IPMI] Intel, Hewlett-Packard, NEC, and Dell, "Intelligent Platform Management Interface Specification, Second Generation", October 2013, <https://www.intel.la/content/dam/www/public/us/en/documents/specification-updates/ipmi-intelligent-platform-mgt-interface-spec-2nd-gen-v2-0-spec-update.pdf>.

[NEW-HORIZONS] Moore, R. C., "Autonomous safeing and fault protection for the New Horizons mission to Pluto", March 2007, <https://www.sciencedirect.com/science/article/pii/S0094576507000604>.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <https://www.rfc-editor.org/info/rfc2578>.

[RFC2982] Kavasseri, R., Ed., "Distributed Management Expression MIB", RFC 2982, DOI 10.17487/RFC2982, October 2000, <https://www.rfc-editor.org/info/rfc2982>.

[RFC3165] Levi, D. and J. Schoenwaelder, "Definitions of Managed Objects for the Delegation of Management Scripts", RFC 3165, DOI 10.17487/RFC3165, August 2001, <https://www.rfc-editor.org/info/rfc3165>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <https://www.rfc-editor.org/info/rfc3410>.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, DOI 10.17487/RFC3411, December 2002, <https://www.rfc-editor.org/info/rfc3411>.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, DOI 10.17487/RFC3414, December 2002, <https://www.rfc-editor.org/info/rfc3414>.

[RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <https://www.rfc-editor.org/info/rfc3416>.

[RFC3417] Presuhn, R., Ed., "Transport Mappings for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3417, DOI 10.17487/RFC3417, December 2002, <https://www.rfc-editor.org/info/rfc3417>.

[RFC3418] Presuhn, R., Ed., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, DOI 10.17487/RFC3418, December 2002, <https://www.rfc-editor.org/info/rfc3418>.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, DOI 10.17487/RFC4838, April 2007, <https://www.rfc-editor.org/info/rfc4838>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC5591] Harrington, D. and W. Hardaker, "Transport Security Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 5591, DOI 10.17487/RFC5591, June 2009, <https://www.rfc-editor.org/info/rfc5591>.

[RFC5592] Harrington, D., Salowey, J., and W. Hardaker, "Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5592, DOI 10.17487/RFC5592, June 2009, <https://www.rfc-editor.org/info/rfc5592>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5953] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", RFC 5953, DOI 10.17487/RFC5953, August 2010, <https://www.rfc-editor.org/info/rfc5953>.

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

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

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <https://www.rfc-editor.org/info/rfc7575>.

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

[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>.

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

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

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

[RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <https://www.rfc-editor.org/info/rfc8294>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

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

[RFC8368] Eckert, T., Ed. and M. Behringer, "Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM)", RFC 8368, DOI 10.17487/RFC8368, May 2018, <https://www.rfc-editor.org/info/rfc8368>.

[RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <https://www.rfc-editor.org/info/rfc8639>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

[RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <https://www.rfc-editor.org/info/rfc8990>.

[RFC8993] Behringer, M., Ed., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", RFC 8993, DOI 10.17487/RFC8993, May 2021, <https://www.rfc-editor.org/info/rfc8993>.

[RFC9171] Burleigh, S., Fall, K., and E. Birrane, III, "Bundle Protocol Version 7", RFC 9171, DOI 10.17487/RFC9171, January 2022, <https://www.rfc-editor.org/info/rfc9171>.

[RFC9172] Birrane, III, E. and K. McKeever, "Bundle Protocol Security (BPSec)", RFC 9172, DOI 10.17487/RFC9172, January 2022, <https://www.rfc-editor.org/info/rfc9172>.

[RFC9254] Veillette, M., Ed., Petrov, I., Ed., Pelov, A., Bormann, C., and M. Richardson, "Encoding of Data Modeled with YANG in the Concise Binary Object Representation (CBOR)", RFC 9254, DOI 10.17487/RFC9254, July 2022, <https://www.rfc-editor.org/info/rfc9254>.

[xml-infoset] World Wide Web Consortium, "XML Information Set (Second Edition)", February 2004, <https://www.w3.org/TR/2004/REC-xml-infoset-20040204/>.

[XPath] World Wide Web Consortium, "XML Path Language (XPath) Version 1.0", November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

Acknowledgements

Brian Sipos of the Johns Hopkins University Applied Physics Laboratory (JHU/APL) provided excellent technical review of the DTNMA concepts presented in this document and additional information related to existing network management techniques.

Authors' Addresses

Edward J. Birrane
Johns Hopkins Applied Physics Laboratory
Email: Edward.Birrane@jhuapl.edu

Sarah E. Heiner
Johns Hopkins Applied Physics Laboratory
Email: Sarah.Heiner@jhuapl.edu

Emery Annis
Johns Hopkins Applied Physics Laboratory
Email: Emery.Annis@jhuapl.edu

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 2 August 2024

R.G. Wilton
Cisco Systems
S. Mansfield
Ericsson

30 January 2024

Common Interface Extension YANG Data Models
draft-ietf-netmod-intf-ext-yang-13

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.

Table of Contents

1. Introduction 3
1.1. Terminology 4
1.2. Tree Diagrams 4
1.3. Prefixes in Data Node Names 4
2. Interface Extensions Module 4
2.1. Link Flap Suppression 6
2.2. Dampening 7
2.2.1. Suppress Threshold 7
2.2.2. Half-Life Period 7
2.2.3. Reuse Threshold 7
2.2.4. Maximum Suppress Time 8
2.3. Encapsulation 8
2.4. Loopback 8
2.5. Maximum frame size 9
2.6. Sub-interface 9
2.7. Forwarding Mode 9
3. Interfaces Ethernet-Like Module 10
4. Interface Extensions YANG Module 10
5. Interfaces Ethernet-Like YANG Module 22
6. Examples 25
6.1. Carrier delay configuration 25
6.2. Dampening configuration 27
6.3. MAC address configuration 27
7. Acknowledgements 29
8. IANA Considerations 29
8.1. YANG Module Registrations 29
9. Security Considerations 30
9.1. ietf-if-extensions.yang 30
9.2. ietf-if-ethernet-like.yang 31
10. References 32
10.1. Normative References 32
10.2. Informative References 32
Authors' Addresses 33

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 behavior.

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 implementers of the YANG module will decide to support all features. Hence, separate feature keywords are defined for each logically discrete feature to allow implementers 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 [RFC2119] [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].

1.3. Prefixes in Data Node Names

In this document, names of data nodes and other data model objects are prefixed using the standard prefix associated with the corresponding YANG imported modules, as shown in Table 1.

Prefix	YANG Module	Reference
if-ext	ietf-if-extensions	This document
ethlike	ietf-if-ethernet-like	This document
yang	ietf-yang-types	[RFC6991]
if	ietf-interfaces	[RFC8343]
ianaift	iana-if-type	[RFC7224]

Table 1: Prefixes for imported YANG modules

2. Interface Extensions Module

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

The module provides:

* A link flap suppression 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 link-flap-suppression {link-flap-suppression}?
    |  +--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. Link Flap Suppression

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

2.2. Dampening

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

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

2.2.1. Suppress Threshold

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

2.2.2. Half-Life Period

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

2.2.3. Reuse Threshold

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

2.2.4. Maximum Suppress Time

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

2.3. Encapsulation

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

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

2.4. Loopback

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

The following loopback modes are defined:

* 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
    +--ro in-discard-overflows?            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@2023-01-26.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) 2023 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.

     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 2023-01-26 {
        description
          "Initial revision.";

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

      feature link-flap-suppression {
        description
          "This feature indicates that configurable interface link
           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 Link Flap Suppression";
      }

      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 link-flap-suppression 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 Link Flap Suppression feature.
       */
      container link-flap-suppression {
        if-feature "link-flap-suppression";
```

```
        description
          "Holds link flap 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 link signal must be continuously
             down before the interface state is brought down. If not
             configured, the behavior on loss of link 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 link signal must be continuously present and error
             free before the interface state is allowed to transition
             from down to up.  If not configured, the behavior 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 link 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 link flap suppression
             timer.";
        }
        leaf timer-running {
          type enumeration {
            enum none {
              description
                "No link flap suppression timer is running.";
            }
            enum up {
```

```
            description
              "link-flap-suppression up timer is running.  The
               underlying link state is up, but interface state is
               not reported as up.";
          }
          enum down {
            description
              "link-flap-suppression down timer is running.
               Interface state is reported as up, but the underlying
               link state is actually down.";
          }
        }
        config false;
        description
          "Reports whether a link flap suppression timer is actively
           running, in which case the interface state does not match
           the underlying link state.";
      }

      reference "RFC XXXX, Section 2.1 Link Flap Suppression";
    }

    /*
     * 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 [IEEE_802.3.2_2019].

```
<CODE BEGINS> file "ietf-if-ethernet-like@2023-01-26.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) 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 2023-01-26 {
       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).";
        }

        leaf in-discard-overflows {
          type yang:counter64;
          units frames;
          description
             "A count of the number of frames discarded because of
              overflows.";
        }
      }
    }
    <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 link flap suppression 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:link-flap-suppression>
      <if-ext:down>0</if-ext:down>
      <if-ext:up>50</if-ext:up>
    </if-ext:link-flap-suppression>
  </interface>
</interfaces>
```

The following example shows explicit link flap suppression 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:link-flap-suppression>
        <if-ext:down>50</if-ext:down>
        <if-ext:up>1000</if-ext:up>
      </if-ext:link-flap-suppression>
    </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. IANA Considerations

8.1. YANG Module Registrations

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

The ietf-if-extensions module:

- Name: ietf-if-extensions
- XML Namespace: urn:ietf:params:xml:ns:yang:ietf-if-extensions
- Prefix: if-ext
- Reference: RFCXXXX

The ietf-if-ethernet-like module:

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

Prefix: ethlike

Reference: RFCXXXX

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

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

Registrant Contact: The IESG.

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

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

Registrant Contact: The IESG.

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

9. 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 8341 [RFC8341] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in 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:

9.1. 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/link-flap-suppression/down

* /if:interfaces/if:interface/link-flap-suppression/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

9.2. 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

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>.

[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>.

10.2. Informative References

[I-D.ietf-netmod-sub-intf-vlan-model] Wilton, R. and S. Mansfield, "Sub-interface VLAN YANG Data Models", Work in Progress, Internet-Draft, draft-ietf-netmod-sub-intf-vlan-model-09, 18 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-sub-intf-vlan-model-09>.

[IEEE_802.3.2_2019]
IEEE, "IEEE Standard for Ethernet – YANG Data Model Definitions", IEEE 802-3, DOI 10.1109/IEEESTD.2019.8737019, 14 June 2019, <https://ieeexplore.ieee.org/document/8737019>.

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

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

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

Authors' Addresses

Robert Wilton
Cisco Systems
Email: rwilton@cisco.com

Scott Mansfield
Ericsson
Email: scott.mansfield@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 2 August 2024

R.G. Wilton, Ed.
Cisco Systems
S. Mansfield, Ed.
Ericsson

30 January 2024

Sub-interface VLAN YANG Data Models
draft-ietf-netmod-sub-intf-vlan-model-10

Abstract

This document defines YANG modules to add support for classifying traffic received on interfaces as Ethernet/VLAN framed packets to sub-interfaces based on the fields available in the Ethernet/VLAN frame headers. These modules allow configuration of Layer 3 and Layer 2 sub-interfaces (e.g. L2VPN attachment circuits) that can interoperate with IETF based forwarding protocols; such as IP and L3VPN services; or L2VPN services like VPWS, VPLS, and EVPN. The sub-interfaces also interoperate with VLAN tagged traffic orignating from an IEEE 802.1Q compliant bridge.

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

The YANG data models in this document conforms to the Network Management Datastore Architecture (NMDA) defined in RFC 8342.

Status of This Memo

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

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

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

This Internet-Draft will expire on 2 August 2024.

Table of Contents

1. Introduction 3
1.1. Terminology 3
1.2. Tree Diagrams 4
1.3. Prefixes in Data Node Names 4
2. Objectives 4
2.1. Interoperability with IEEE 802.1Q compliant bridges . . . 5
3. Interface VLAN Encapsulation Model 5
4. Interface Flexible Encapsulation Model 5
5. VLAN Encapsulation YANG Module 8
6. Flexible Encapsulation YANG Module 11
7. Examples 22
7.1. Layer 3 sub-interfaces with IPv6 22
7.2. Layer 2 sub-interfaces with L2VPN 24
8. Acknowledgements 26
9. IANA Considerations 26
9.1. YANG Module Registrations 26
10. Security Considerations 27
10.1. ietf-if-vlan-encapsulation.yang 28
10.2. ietf-if-flexible-encapsulation.yang 28
11. References 30
11.1. Normative References 30
11.2. Informative References 31
Appendix A. Comparison with the IEEE 802.1Q Configuration Model 32
A.1. Sub-interface based configuration model overview 32
A.2. IEEE 802.1Q Bridge Configuration Model Overview 33
A.3. Possible Overlap Between the Two Models 34
Authors' Addresses 34

1. Introduction

This document defines two YANG [RFC7950] modules that augment the encapsulation choice YANG element defined in Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang] and the generic interfaces data model defined in [RFC8343]. The two modules provide configuration nodes to support classification of Ethernet/VLAN traffic to sub-interfaces, that can have interface based feature and service configuration applied to them.

The purpose of these models is to allow IETF defined forwarding protocols, for example, IPv6 [RFC8200], Ethernet Pseudo Wires [RFC4448] and VPLS [RFC4761] [RFC4762], when configured via appropriate YANG data models [RFC8344] [I-D.ietf-bess-l2vpn-yang], to interoperate with VLAN tagged traffic received from an IEEE 802.1Q compliant bridge.

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

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

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

ietf-if-vlan-encapsulation.yang - Defines the model for basic classification of VLAN tagged traffic, normally to L3 packet forwarding services

ietf-if-flexible-encapsulation.yang - Defines the model for flexible classification of Ethernet/VLAN traffic, normally to L2 frame forwarding services

1.1. Terminology

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

The term 'sub-interface' is defined in section 2.6 of Interface Extensions YANG [I-D.ietf-netmod-intf-ext-yang].

1.2. Tree Diagrams

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

1.3. Prefixes in Data Node Names

In this document, names of data nodes and other data model objects are prefixed using the standard prefix associated with the corresponding YANG imported modules, as shown in Table 1.

Prefix	YANG Module	Reference
if-vlan	ietf-if-vlan-encapsulation	This document
if-flex	ietf-if-flexible-encapsulation	This document
if-ext	ietf-if-extensions	[I-D.ietf-netmod-intf-ext-yang]
if	ietf-interfaces	[RFC8343]
ianaift	iana-if-type	[RFC7224]
dot1q-types	ieee802-dot1q-types	[IEEE_802.1Q_2022]

Table 1: Prefixes for imported YANG modules

2. Objectives

The primary aim of the YANG modules contained in this draft is to provide the core model that is required to implement VLAN transport services on router based devices that is fully compatible with IEEE 802.1Q compliant bridges.

A secondary aim is for the modules to be structured in such a way that they can be cleanly extended in future.

2.1. Interoperability with IEEE 802.1Q compliant bridges

The modules defined in this document are designed to fully interoperate with IEEE 802.1Q compliant bridges. In particular, the models are restricted to only matching, adding, or rewriting the 802.1Q VLAN tags in frames in ways that are compatible with IEEE 802.1Q compliant bridges.

3. Interface VLAN Encapsulation Model

The Interface VLAN encapsulation model provides appropriate leaves for termination of an 802.1Q VLAN tagged segment to a sub-interface (or interface) based L3 service, such as IP. It allows for termination of traffic with one or two 802.1Q VLAN tags.

The L3 service must be configured via a separate YANG data model, e.g., [RFC8344]. A short example of configuring 802.1Q VLAN sub-interfaces with IP using YANG is provided in Section 7.1.

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

```
module: ietf-if-vlan-encapsulation
  augment /if:interfaces/if:interface/if-ext:encapsulation
            /if-ext:encaps-type:
    +--:(dot1q-vlan)
       +--rw dot1q-vlan
          +--rw outer-tag
          |  +--rw tag-type dot1q-tag-type
          |  +--rw vlan-id     vlanid
          +--rw second-tag!
             +--rw tag-type dot1q-tag-type
             +--rw vlan-id     vlanid
```

4. Interface Flexible Encapsulation Model

The Interface Flexible Encapsulation model is designed to allow for the flexible provisioning of layer 2 services. It provides the capability to classify and demultiplex Ethernet/VLAN frames received on an Ethernet trunk interface to sub-interfaces based on the fields available in the layer 2 headers. Once classified to sub-interfaces, it provides the capability to selectively modify fields within the layer 2 frame header before the frame is handed off to the appropriate forwarding code for further handling. The forwarding instance, e.g., L2VPN, VPLS, etc., is configured using a separate YANG configuration model defined elsewhere, e.g.,

[I-D.ietf-bess-l2vpn-yang].

The model supports a common core set of layer 2 header matches based on the 802.1Q tag type and VLAN Ids contained within the header up to a tag stack depth of two tags.

The model supports flexible rewrites of the layer 2 frame header for data frames as they are processed on the interface. It defines a set of standard tag manipulations that allow for the insertion, removal, or rewrite of one or two 802.1Q VLAN tags. The expectation is that manipulations are generally implemented in a symmetrical fashion, i.e. if a manipulation is performed on ingress traffic on an interface then the reverse manipulation is always performed on egress traffic out of the same interface. However, the model also allows for asymmetrical rewrites, which may be required to implement some forwarding models (such as E-Tree).

The model also allows a flexible encapsulation and rewrite to be configured directly on an Ethernet or LAG interface without configuring separate child sub-interfaces. Ingress frames that do not match the encapsulation are dropped. Egress frames MUST conform to the encapsulation.

The final aim for the model design is for it to be cleanly extensible to add in additional match and rewrite criteria of the layer 2 header, such as matching on the source or destination MAC address, PCP or DEI fields in the 802.1Q tags, or the EtherType of the frame payload. Rewrites can also be extended to allow for modification of other fields within the layer 2 frame header.

A short example of configuring 802.1Q VLAN sub-interfaces with L2VPN using YANG is provided in Section 7.2.

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

```
module: ietf-if-flexible-encapsulation
  augment /if:interfaces/if:interface/if-ext:encapsulation
            /if-ext:encaps-type:
    +--:(flexible)
       +--rw flexible
          +--rw match
          |  +--rw (match-type)
          |     +--:(default)
          |     |  +--rw default?                empty
          |     +--:(untagged)
          |     |  +--rw untagged?               empty
```

```
|      +--:(dot1q-priority-tagged)
|      |  +--rw dot1q-priority-tagged
|      |     +--rw tag-type dot1q-types:dot1q-tag-type
|      +--:(dot1q-vlan-tagged)
|         +--rw dot1q-vlan-tagged
|            +--rw outer-tag
|            |  +--rw tag-type dot1q-tag-type
|            |  +--rw vlan-id     union
|            +--rw second-tag!
|            |  +--rw tag-type dot1q-tag-type
|            |  +--rw vlan-id     union
|            +--rw match-exact-tags?   empty
|
+--rw rewrite {flexible-rewrites}?
|  +--rw (direction)?
|     +--:(symmetrical)
|     |  +--rw symmetrical
|     |     +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|     |        +--rw pop-tags?    uint8
|     |        +--rw push-tags!
|     |           +--rw outer-tag
|     |           |  +--rw tag-type dot1q-tag-type
|     |           |  +--rw vlan-id     vlanid
|     |           +--rw second-tag!
|     |              +--rw tag-type dot1q-tag-type
|     |              +--rw vlan-id     vlanid
|     +--:(asymmetrical) {asymmetric-rewrites}?
|        +--rw ingress
|        |  +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|        |     +--rw pop-tags?    uint8
|        |     +--rw push-tags!
|        |        +--rw outer-tag
|        |        |  +--rw tag-type dot1q-tag-type
|        |        |  +--rw vlan-id     vlanid
|        |        +--rw second-tag!
|        |           +--rw tag-type dot1q-tag-type
|        |           +--rw vlan-id     vlanid
|        +--rw egress
|           +--rw dot1q-tag-rewrite {dot1q-tag-rewrites}?
|              +--rw pop-tags?    uint8
|              +--rw push-tags!
|                 +--rw outer-tag
|                 |  +--rw tag-type dot1q-tag-type
|                 |  +--rw vlan-id     vlanid
|                 +--rw second-tag!
|                    +--rw tag-type dot1q-tag-type
|                    +--rw vlan-id     vlanid
|
+--rw local-traffic-default-encaps!
   +--rw outer-tag
```

```
               |  +--rw tag-type dot1q-tag-type
               |  +--rw vlan-id     vlanid
               +--rw second-tag!
                  +--rw tag-type dot1q-tag-type
                  +--rw vlan-id     vlanid
```

5. VLAN Encapsulation YANG Module

This YANG module augments the 'encapsulation' container defined in ietf-if-extensions.yang [I-D.ietf-netmod-intf-ext-yang]. It also contains references to [RFC8343], [RFC7224], and [IEEE_802.1Q_2022].

<CODE BEGINS> file "ietf-if-vlan-encapsulation@2023-01-26.yang"

```
module ietf-if-vlan-encapsulation {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation";
  prefix if-vlan;

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

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

  import ieee802-dot1q-types {
    prefix dot1q-types;
    revision-date 2022-01-19;
    reference
      "IEEE Std 802.1Q-2022: IEEE Standard for Local and
       metropolitan area networks -- Bridges and Bridged Networks";
  }

  import ietf-if-extensions {
    prefix if-ext;
    reference
      "RFC XXXX: Common Interface Extension YANG Data Models";
  }

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

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

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

     description
       "This YANG module models configuration to classify IEEE 802.1Q
        VLAN tagged Ethernet traffic by exactly matching the tag type
        and VLAN identifier of one or two 802.1Q VLAN tags in the frame.

        Copyright (c) 2023 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.

        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 2023-01-26 {
       description
         "Latest draft revision";
       reference
         "RFC XXXX: Sub-interface VLAN YANG Data Models";
     }

     augment "/if:interfaces/if:interface/if-ext:encapsulation/"
           + "if-ext:encaps-type" {
       when "derived-from-or-self(../if:type,
                                  'ianaift:ethernetCsmacd') or
             derived-from-or-self(../if:type,
                                  'ianaift:ieee8023adLag') or
             derived-from-or-self(../if:type, 'ianaift:l2vlan') or
             derived-from-or-self(../if:type,
                                  'if-ext:ethSubInterface')" {
```

```
          description
            "Applies only to Ethernet-like interfaces and
             sub-interfaces.";
        }

        description
          "Augment the generic interface encapsulation with basic 802.1Q
           VLAN tag classifications";

        case dot1q-vlan {
          container dot1q-vlan {

            description
              "Classifies 802.1Q VLAN tagged Ethernet frames to a
               sub-interface (or interface) by exactly matching the
               number of tags, tag type(s) and VLAN identifier(s).

               Only frames matching the classification configured on a
               sub-interface/interface are processed on that
               sub-interface/interface.

               Frames that do not match any sub-interface are processed
               directly on the parent interface, if it is associated with
               a forwarding instance, otherwise they are dropped.";

            container outer-tag {
              must 'tag-type = "dot1q-types:s-vlan" or '
                 + 'tag-type = "dot1q-types:c-vlan"' {

                error-message
                  "Only C-VLAN and S-VLAN tags can be matched.";

                description
                  "For IEEE 802.1Q interoperability, only C-VLAN and
                   S-VLAN tags are matched.";
              }

              description
                "Specifies the VLAN tag values to match against the
                 outermost (first) 802.1Q VLAN tag in the frame.";

              uses dot1q-types:dot1q-tag-classifier-grouping;
            }

            container second-tag {
              must '../outer-tag/tag-type = "dot1q-types:s-vlan" and '
                 + 'tag-type = "dot1q-types:c-vlan"' {
```

```
              error-message
                "When matching two 802.1Q VLAN tags, the outermost
                 (first) tag in the frame MUST be specified and be of
                 S-VLAN type and the second tag in the frame must be of
                 C-VLAN tag type.";

              description
                "For IEEE 802.1Q interoperability, when matching two
                 802.1Q VLAN tags, it is REQUIRED that the outermost
                 tag exists and is an S-VLAN, and the second tag is a
                 C-VLAN.";
            }

            presence "Classify frames that have two 802.1Q VLAN tags.";

            description
              "Specifies the VLAN tag values to match against the
               second outermost 802.1Q VLAN tag in the frame.";

            uses dot1q-types:dot1q-tag-classifier-grouping;
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Flexible Encapsulation YANG Module

This YANG module augments the 'encapsulation' container defined in ietf-if-extensions.yang [I-D.ietf-netmod-intf-ext-yang]. This YANG module also augments the 'interface' list entry defined in [RFC8343]. It also contains references to [RFC7224], and [IEEE_802.1Q_2022].

```
<CODE BEGINS> file "ietf-if-flexible-encapsulation@2023-01-26.yang"
module ietf-if-flexible-encapsulation {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation";
  prefix if-flex;

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

  import iana-if-type {
```

```
      prefix ianaift;
      reference
        "RFC 7224: IANA Interface Type YANG Module";
    }

    import ieee802-dot1q-types {
      prefix dot1q-types;
      revision-date 2022-01-19;
      reference
        "IEEE Std 802.1Q-2022: IEEE Standard for Local and
         metropolitan area networks -- Bridges and Bridged Networks";
    }

    import ietf-if-extensions {
      prefix if-ext;
      reference
        "RFC XXXX: Common Interface Extension YANG Data Models";
    }

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

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

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

    description
      "This YANG module describes interface configuration for flexible
       classification and rewrites of IEEE 802.1Q VLAN tagged Ethernet
       traffic.

       Copyright (c) 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.
```

```
        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 2023-01-26 {
      description
        "Latest draft revision";
      reference
        "RFC XXXX: Sub-interface VLAN YANG Data Models";
    }

    feature flexible-rewrites {
      description
        "This feature indicates that the network element supports
          specifying flexible rewrite operations.";
    }

    feature asymmetric-rewrites {
      description
        "This feature indicates that the network element supports
         specifying different rewrite operations for the ingress
         rewrite operation and egress rewrite operation.";
    }

    feature dot1q-tag-rewrites {
      description
        "This feature indicates that the network element supports the
         flexible rewrite functionality specifying 802.1Q tag
         rewrites.";
    }

    grouping flexible-match {
      description
        "Represents a flexible frame classification:

         The rules for a flexible match are:
           1. Match-type: default, untagged, priority tag, or tag
              stack.
           2. Each tag in the stack of tags matches:
            a. tag type (802.1Q or 802.1ad) +
            b. tag value:
              i.   single tag
              ii.  set of tag ranges/values.
              iii. 'any' keyword";

      choice match-type {
```

```
        mandatory true;

        description
          "Provides a choice of how the frames may be
           matched";

        case default {
          description
            "Default match";

          leaf default {
            type empty;

            description
              "Default match.  Matches all traffic not matched to any
               other peer sub-interface by a more specific
               encapsulation.";
          }
        }

        case untagged {
          description
            "Match untagged Ethernet frames only";

          leaf untagged {
            type empty;

            description
              "Untagged match.  Matches all untagged traffic.";
          }
        }

        case dot1q-priority-tagged {
          description
            "Match 802.1Q priority tagged Ethernet frames only";

          container dot1q-priority-tagged {
            description
              "802.1Q priority tag match";

            leaf tag-type {
              type dot1q-types:dot1q-tag-type;
              mandatory true;

              description
                "The 802.1Q tag type of matched priority
                 tagged packets";
            }
```

```
          }
        }

        case dot1q-vlan-tagged {
          container dot1q-vlan-tagged {
            description
              "Matches VLAN tagged frames";

            container outer-tag {
              must 'tag-type = "dot1q-types:s-vlan" or '
                 + 'tag-type = "dot1q-types:c-vlan"' {

                error-message
                  "Only C-VLAN and S-VLAN tags can be matched.";

                description
                  "For IEEE 802.1Q interoperability, only C-VLAN and
                   S-VLAN tags can be matched.";
              }

              description
                "Classifies traffic using the outermost (first) VLAN
                 tag on the frame.";

              uses "dot1q-types:"
                 + "dot1q-tag-ranges-or-any-classifier-grouping";
            }

            container second-tag {
              must
                '../outer-tag/tag-type = "dot1q-types:s-vlan" and '
              + 'tag-type = "dot1q-types:c-vlan"' {

                error-message
                  "When matching two tags, the outermost (first) tag
                   must be specified and of S-VLAN type and the second
                   outermost tag must be of C-VLAN tag type.";

                description
                  "For IEEE 802.1Q interoperability, when matching two
                   tags, it is required that the outermost (first) tag
                   exists and is an S-VLAN, and the second outermost
                   tag is a C-VLAN.";
              }

              presence "Also classify on the second VLAN tag.";

              description
```

```
                "Classifies traffic using the second outermost VLAN tag
                 on the frame.";

              uses "dot1q-types:"
                 + "dot1q-tag-ranges-or-any-classifier-grouping";
            }

            leaf match-exact-tags {
              type empty;
              description
                "If set, indicates that all 802.1Q VLAN tags in the
                 Ethernet frame header must be explicitly matched, i.e.
                 the EtherType following the matched tags must not be a
                 802.1Q tag EtherType.  If unset then extra 802.1Q VLAN
                 tags are allowed.";
            }
          }
        }
      }
    }

    grouping dot1q-tag-rewrite {
      description
        "Flexible rewrite grouping.  Can be either be expressed
         symmetrically, or independently in the ingress and/or egress
         directions.";

      leaf pop-tags {
        type uint8 {
          range "1..2";
        }

        description
          "The number of 802.1Q VLAN tags to pop, or translate if used
           in conjunction with push-tags.

           Popped tags are the outermost tags on the frame.";
      }

      container push-tags {
        presence "802.1Q tags are pushed or translated";

        description
          "The 802.1Q tags to push on the front of the frame, or
           translate if configured in conjunction with pop-tags.";

        container outer-tag {
          must 'tag-type = "dot1q-types:s-vlan" or '
```

```
              + 'tag-type = "dot1q-types:c-vlan"' {

            error-message "Only C-VLAN and S-VLAN tags can be pushed.";

            description
              "For IEEE 802.1Q interoperability, only C-VLAN and S-VLAN
               tags can be pushed.";
          }

          description
            "The outermost (first) VLAN tag to push onto the frame.";

          uses dot1q-types:dot1q-tag-classifier-grouping;
        }

        container second-tag {
          must '../outer-tag/tag-type = "dot1q-types:s-vlan" and '
             + 'tag-type = "dot1q-types:c-vlan"' {

            error-message
              "When pushing/rewriting two tags, the outermost tag must
               be specified and of S-VLAN type and the second outermost
               tag must be of C-VLAN tag type.";

            description
              "For IEEE 802.1Q interoperability, when pushing two tags,
               it is required that the outermost tag exists and is an
               S-VLAN, and the second outermost tag is a C-VLAN.";
          }

          presence
            "In addition to the first tag, also push/rewrite a second
             VLAN tag.";

          description
            "The second outermost VLAN tag to push onto the frame.";

          uses dot1q-types:dot1q-tag-classifier-grouping;
        }
      }
    }

    grouping flexible-rewrite {
      description
        "Grouping for flexible rewrites of fields in the L2 header.

         Restricted to flexible 802.1Q VLAN tag rewrites, but could be
         extended to cover rewrites of other fields in the L2 header in
```

```
          future.";

        container dot1q-tag-rewrite {
          if-feature "dot1q-tag-rewrites";

          description
            "802.1Q VLAN tag rewrite.

             Translate operations are expressed as a combination of tag
             push and pop operations.  E.g., translating the outer tag is
             expressed as popping a single tag, and pushing a single tag.
             802.1Q tags that are translated SHOULD preserve the PCP and
             DEI fields unless if a different QoS behavior has been
             specified.";
          uses dot1q-tag-rewrite;
        }
      }

      augment "/if:interfaces/if:interface/if-ext:encapsulation/"
            + "if-ext:encaps-type" {
        when "derived-from-or-self(../if:type,
                                   'ianaift:ethernetCsmacd') or
              derived-from-or-self(../if:type,
                                   'ianaift:ieee8023adLag') or
              derived-from-or-self(../if:type, 'ianaift:l2vlan') or
              derived-from-or-self(../if:type,
                                   'if-ext:ethSubInterface')" {

          description
            "Applies only to Ethernet-like interfaces and
             sub-interfaces.";
        }

        description
          "Augment the generic interface encapsulation with flexible
           match and rewrite for VLAN sub-interfaces.";

        case flexible {
          description
            "Flexible encapsulation and rewrite";

          container flexible {
            description
              "Flexible encapsulation allows for the matching of ranges
               and sets of 802.1Q VLAN Tags and performing rewrite
               operations on the VLAN tags.

               The structure is also designed to be extended to allow for
```

```
           matching/rewriting other fields within the L2 frame header
           if required.";

         container match {
           description
             "Flexibly classifies Ethernet frames to a sub-interface
              (or interface) based on the L2 header fields.

              Only frames matching the classification configured on a
              sub-interface/interface are processed on that
              sub-interface/interface.

              Frames that do not match any sub-interface are processed
              directly on the parent interface, if it is associated
              with a forwarding instance, otherwise they are dropped.

              If a frame could be classified to multiple
              sub-interfaces then they get classified to the
              sub-interface with the most specific match.  E.g.,
              matching two VLAN tags in the frame is more specific
              than matching the outermost VLAN tag, which is more
              specific than the catch all 'default' match.";

           uses flexible-match;
         }

         container rewrite {
           if-feature "flexible-rewrites";

           description
             "L2 frame rewrite operations.

              Rewrites allows for modifications to the L2 frame header
              as it transits the interface/sub-interface.  Examples
              include adding a VLAN tag, removing a VLAN tag, or
              rewriting the VLAN Id carried in a VLAN tag.";

           choice direction {
             description
               "Whether the rewrite policy is symmetrical or
                asymmetrical.";

             case symmetrical {
               container symmetrical {
                 uses flexible-rewrite;

                 description
                   "Symmetrical rewrite.  Expressed in the ingress
```

```
                    direction, but the reverse operation is applied to
                    egress traffic.

                    E.g., if a tag is pushed on ingress traffic, then
                    the reverse operation is a 'pop 1', that is
                    performed on traffic egressing the interface, so
                    a peer device sees a consistent L2 encapsulation
                    for both ingress and egress traffic.";
                }
              }

              case asymmetrical {
                if-feature "asymmetric-rewrites";

                description
                  "Asymmetrical rewrite.

                   Rewrite operations may be specified in only a single
                   direction, or different rewrite operations may be
                   specified in each direction.";

                container ingress {
                  uses flexible-rewrite;

                  description
                    "A rewrite operation that only applies to ingress
                     traffic.

                     Ingress rewrite operations are performed before
                     the frame is subsequently processed by the
                     forwarding operation.";
                }

                container egress {
                  uses flexible-rewrite;

                  description
                    "A rewrite operation that only applies to egress
                     traffic.";
                }
              }
            }
          }

          container local-traffic-default-encaps {
            presence "A local traffic default encapsulation has been
                      specified.";
```

```
            description
              "Specifies the 802.1Q VLAN tags to use by default for
               locally sourced traffic from the interface.

               Used for encapsulations that match a range of VLANs (or
               'any'), where the source VLAN Ids are otherwise
               ambiguous.";

            container outer-tag {
              must 'tag-type = "dot1q-types:s-vlan" or '
                 + 'tag-type = "dot1q-types:c-vlan"' {

                error-message
                  "Only C-VLAN and S-VLAN tags can be matched.";

                description
                  "For IEEE 802.1Q interoperability, only C-VLAN and
                   S-VLAN tags can be matched.";
              }

              description
                "The outermost (first) VLAN tag for locally sourced
                 traffic.";

              uses dot1q-types:dot1q-tag-classifier-grouping;
            }

            container second-tag {
              must
                '../outer-tag/tag-type = "dot1q-types:s-vlan" and '
              + 'tag-type = "dot1q-types:c-vlan"' {

                error-message
                  "When specifying two tags, the outermost (first) tag
                   must be specified and of S-VLAN type and the second
                   outermost tag must be of C-VLAN tag type.";

                description
                  "For IEEE 802.1Q interoperability, when specifying
                   two tags, it is required that the outermost (first)
                   tag exists and is an S-VLAN, and the second
                   outermost tag is a C-VLAN.";
              }

              presence
                "Indicates existence of a second outermost VLAN tag.";

              description
```

```
                "The second outermost VLAN tag for locally sourced
                 traffic.";

              uses dot1q-types:dot1q-tag-classifier-grouping;
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

7. Examples

The following sections give examples of configuring a sub-interface supporting L3 forwarding, and a sub-interface being used in conjunction with the IETF L2VPN YANG model [I-D.ietf-bess-l2vpn-yang].

7.1. Layer 3 sub-interfaces with IPv6

This example illustrates two layer sub-interfaces, 'eth0.1' and 'eth0.2', both are child interfaces of the Ethernet interface 'eth0'.

'eth0.1' is configured to match traffic with two VLAN tags: an outer S-VLAN of 10 and an inner C-VLAN of 20.

'eth0.2' is configured to match traffic with a single S-VLAN tag, with VLAN Id 11.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
  xmlns:dot1q-types="urn:ieee:std:802.1Q:yang:ieee802-dot1q-types"
  xmlns:if-ext="urn:ietf:params:xml:ns:yang:ietf-if-extensions">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
    </interface>
    <interface>
      <name>eth0.1</name>
      <type>ianaift:l2vlan</type>
      <if-ext:parent-interface>eth0</if-ext:parent-interface>
      <if-ext:encapsulation>
        <dot1q-vlan
```

```
            xmlns=
               "urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation">
             <outer-tag>
               <tag-type>dot1q-types:s-vlan</tag-type>
               <vlan-id>10</vlan-id>
             </outer-tag>
             <second-tag>
               <tag-type>dot1q-types:c-vlan</tag-type>
               <vlan-id>20</vlan-id>
             </second-tag>
           </dot1q-vlan>
         </if-ext:encapsulation>
         <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
           <forwarding>true</forwarding>
           <address>
             <ip>2001:db8:10::1</ip>
             <prefix-length>48</prefix-length>
           </address>
         </ipv6>
       </interface>
       <interface>
         <name>eth0.2</name>
         <type>ianaift:l2vlan</type>
         <if-ext:parent-interface>eth0</if-ext:parent-interface>
         <if-ext:encapsulation>
           <dot1q-vlan
            xmlns=
               "urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation">
             <outer-tag>
               <tag-type>dot1q-types:s-vlan</tag-type>
               <vlan-id>11</vlan-id>
             </outer-tag>
           </dot1q-vlan>
         </if-ext:encapsulation>
         <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
           <forwarding>true</forwarding>
           <address>
             <ip>2001:db8:11::1</ip>
             <prefix-length>48</prefix-length>
           </address>
         </ipv6>
       </interface>
     </interfaces>
   </config>
```

7.2. Layer 2 sub-interfaces with L2VPN

This example illustrates a layer 2 sub-interface 'eth0.3' configured to match traffic with a S-VLAN tag of 10, and C-VLAN tag of 21; and remov the outer tag (S-VLAN 10) before the traffic is passed off to the L2VPN service.

It also illustrates another sub-interface 'eth1.0' under a separate physical interface configured to match traffic with a C-VLAN of 50, with the tag removed before traffic is given to any service. Sub-interface 'eth1.0' is not currently bound to any service and hence traffic classified to that sub-interface is dropped.

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <interfaces
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
  xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type"
  xmlns:dot1q-types="urn:ieee:std:802.1Q:yang:ieee802-dot1q-types"
  xmlns:if-ext="urn:ietf:params:xml:ns:yang:ietf-if-extensions">
    <interface>
      <name>eth0</name>
      <type>ianaift:ethernetCsmacd</type>
    </interface>
    <interface>
      <name>eth0.3</name>
      <type>ianaift:l2vlan</type>
      <if-ext:parent-interface>eth0</if-ext:parent-interface>
      <if-ext:encapsulation>
        <flexible xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation">
          <match>
            <dot1q-vlan-tagged>
              <outer-tag>
                <tag-type>dot1q-types:s-vlan</tag-type>
                <vlan-id>10</vlan-id>
              </outer-tag>
              <second-tag>
                <tag-type>dot1q-types:c-vlan</tag-type>
                <vlan-id>21</vlan-id>
              </second-tag>
            </dot1q-vlan-tagged>
          </match>
          <rewrite>
            <symmetrical>
              <dot1q-tag-rewrite>
                <pop-tags>1</pop-tags>
```

```
                </dot1q-tag-rewrite>
              </symmetrical>
            </rewrite>
          </flexible>
        </if-ext:encapsulation>
      </interface>
      <interface>
        <name>eth1</name>
        <type>ianaift:ethernetCsmacd</type>
      </interface>
      <interface>
        <name>eth1.0</name>
        <type>ianaift:l2vlan</type>
        <if-ext:parent-interface>eth0</if-ext:parent-interface>
        <if-ext:encapsulation>
          <flexible xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation">
            <match>
              <dot1q-vlan-tagged>
                <outer-tag>
                  <tag-type>dot1q-types:c-vlan</tag-type>
                  <vlan-id>50</vlan-id>
                </outer-tag>
              </dot1q-vlan-tagged>
            </match>
            <rewrite>
              <symmetrical>
                <dot1q-tag-rewrite>
                  <pop-tags>1</pop-tags>
                </dot1q-tag-rewrite>
              </symmetrical>
            </rewrite>
          </flexible>
        </if-ext:encapsulation>
      </interface>
    </interfaces>
    <network-instances
        xmlns="urn:ietf:params:xml:ns:yang:ietf-network-instance">
      <network-instance
       xmlns:l2vpn="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
        <name>p2p-l2-1</name>
        <description>Point to point L2 service</description>
        <l2vpn:type>l2vpn:vpws-instance-type</l2vpn:type>
        <l2vpn:signaling-type>
          l2vpn:ldp-signaling
        </l2vpn:signaling-type>
        <endpoint xmlns="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
          <name>local</name>
```

```
          <ac>
            <name>eth0.3</name>
          </ac>
        </endpoint>
        <endpoint xmlns="urn:ietf:params:xml:ns:yang:ietf-l2vpn">
          <name>remote</name>
          <pw>
            <name>pw1</name>
          </pw>
        </endpoint>
        <vsi-root>
        <!-- Does not Validate -->
        </vsi-root>
      </network-instance>
    </network-instances>
    <pseudowires
        xmlns="urn:ietf:params:xml:ns:yang:ietf-pseudowires">
      <pseudowire>
        <name>pw1</name>
          <peer-ip>2001:db8::50></peer-ip>
          <pw-id>100</pw-id>
      </pseudowire>
    </pseudowires>
  </config>
```

8. Acknowledgements

The authors would particularly like to thank Benoit Claise, John Messenger, Glenn Parsons, and Dan Romascanu for their help progressing this draft.

The authors would also like to thank Martin Bjorklund, Alex Campbell, Don Fedyk, Eric Gray, Giles Heron, Marc Holness, Iftekhar Hussain, Neil Ketley, William Lupton, John Messenger, Glenn Parsons, Ludwig Pauwels, Joseph White, Vladimir Vassilev, and members of the IEEE 802.1 WG for their helpful reviews and feedback on this draft.

9. IANA Considerations

9.1. YANG Module Registrations

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

The ietf-if-vlan-encapsulation module:

Name: ietf-if-vlan-encapsulation

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation

Prefix: if-vlan

Reference: RFCXXXX

The ietf-if-flexible-encapsulation module:

Name: ietf-if-flexible-encapsulation

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation

Prefix: if-flex

Reference: RFCXXXX

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

URI: urn:ietf:params:xml:ns:yang:ietf-if-vlan-encapsulation

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-if-flexible-encapsulation

Registrant Contact: The IESG.

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

10. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH [RFC6242] The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in 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-vlan-encapsulation.yang

The nodes in the vlan encapsulation YANG module are concerned with matching particular frames received on the network device to connect them to a layer 3 forwarding instance, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/dot1q-vlan, that are sensitive to this are:

* outer-tag/tag-type
* outer-tag/vlan-id
* second-tag/tag-type
* second-tag/vlan-id

10.2. ietf-if-flexible-encapsulation.yang

There are many nodes in the flexible encapsulation YANG module that are concerned with matching particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be lost because it is not being classified correctly, or is being classified to a separate sub-interface. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/match, that are sensitive to this are:

* default
* untagged
* dot1q-priority-tagged
* dot1q-priority-tagged/tag-type
* dot1q-vlan-tagged/outer-tag/vlan-type
* dot1q-vlan-tagged/outer-tag/vlan-id
* dot1q-vlan-tagged/second-tag/vlan-type

* dot1q-vlan-tagged/second-tag/vlan-id

There are also many modes in the flexible encapsulation YANG module that are concerned with rewriting the fields in the L2 header for particular frames received on the network device, and as such adding/modifying/deleting these nodes has a high risk of causing traffic to be dropped or incorrectly processed on peer network devices, or it could cause layer 2 tunnels to go down due to a mismatch in negotiated MTU. The nodes, all under the subtree /interfaces/interface/encapsulation/flexible/rewrite, that are sensitive to this are:

* symmetrical/dot1q-tag-rewrite/pop-tags

* symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/tag-type

* symmetrical/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id

* symmetrical/dot1q-tag-rewrite/push-tags/second-tag/tag-type

* symmetrical/dot1q-tag-rewrite/push-tags/second-tag/vlan-id

* asymmetrical/ingress/dot1q-tag-rewrite/pop-tags

* asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type

* asymmetrical/ingress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id

* asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/tag-type

* asymmetrical/ingress/dot1q-tag-rewrite/push-tags/second-tag/vlan-id

* asymmetrical/egress/dot1q-tag-rewrite/pop-tags

* asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/tag-type

* asymmetrical/egress/dot1q-tag-rewrite/push-tags/outer-tag/vlan-id

* asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/tag-type

* asymmetrical/egress/dot1q-tag-rewrite/push-tags/second-tag/vlan-id

Nodes in the flexible-encapsulation YANG module that are concerned with the VLAN tags to use for traffic sourced from the network element could cause protocol sessions (such as CFM) to fail if they are added, modified or deleted. The nodes, all under the subtree /interfaces/interface/flexible-encapsulation/local-traffic-default-encaps that are sensitive to this are:

* outer-tag/vlan-type

* outer-tag/vlan-id

* second-tag/vlan-type

* second-tag/vlan-id

11. References

11.1. Normative References

[I-D.ietf-netmod-intf-ext-yang]
Wilton, R. and S. Mansfield, "Common Interface Extension YANG Data Models", Work in Progress, Internet-Draft, draft-ietf-netmod-intf-ext-yang-12, 18 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-intf-ext-yang-12>.

[IEEE_802.1Q_2022]
IEEE, "IEEE Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks", IEEE 802-1q-2022, DOI 10.1109/IEEESTD.2022.10004498, 30 December 2022, <https://ieeexplore.ieee.org/document/10004498>.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <https://www.rfc-editor.org/info/rfc7224>.

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

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

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <https://www.rfc-editor.org/info/rfc8344>.

11.2. Informative References

[I-D.ietf-bess-l2vpn-yang] Shah, H. C., Brissette, P., Chen, I., Hussain, I., Wen, B., and K. Tiruveedhula, "YANG Data Model for MPLS-based L2VPN", Work in Progress, Internet-Draft, draft-ietf-bess-l2vpn-yang-10, 2 July 2019, <https://datatracker.ietf.org/doc/html/draft-ietf-bess-l2vpn-yang-10>.

[RFC4448] Martini, L., Ed., Rosen, E., El-Aawar, N., and G. Heron, "Encapsulation Methods for Transport of Ethernet over MPLS Networks", RFC 4448, DOI 10.17487/RFC4448, April 2006, <https://www.rfc-editor.org/info/rfc4448>.

[RFC4761] Kompella, K., Ed. and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <https://www.rfc-editor.org/info/rfc4761>.

[RFC4762] Lasserre, M., Ed. and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <https://www.rfc-editor.org/info/rfc4762>.

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

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

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[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>.

Appendix A. Comparison with the IEEE 802.1Q Configuration Model

In addition to the sub-interface based YANG model proposed here, the IEEE 802.1Q working group has developed a YANG model for the configuration of 802.1Q VLANs. This raises the valid question as to whether the models overlap and whether it is necessary or beneficial to have two different models for superficially similar constructs. This section aims to answer that question by summarizing and comparing the two models.

A.1. Sub-interface based configuration model overview

The key features of the sub-interface based configuration model can be summarized as:

* The model is primarily designed to enable layer 2 and layer 3 services on Ethernet interfaces that can be defined in a very flexible way to meet the varied requirements of service providers.

* Traffic is classified from an Ethernet-like interface to sub-interfaces based on fields in the layer 2 header. This is often based on VLAN Ids contained in the frame, but the model is extensible to other arbitrary fields in the frame header.

* Sub-interfaces are just a type of if:interface and hence support any feature configuration YANG models that can be applied generally to interfaces. For example, QoS or ACL models that reference if:interface can be applied to the sub-interfaces, or the sub-interface can be used as an Access Circuit in L2VPN or L3VPN models that reference if:interface.

* In the sub-interface based configuration model, the classification of traffic arriving on an interface to a given sub-interface, based on fields in the layer 2 header, is completely independent of how the traffic is forwarded. The sub-interface can be referenced (via references to if:interface) by other models that specify how traffic is forwarded; thus sub-interfaces can support multiple different forwarding paradigms, including but not limited to: layer 3 (IPv4/IPv6), layer 2 pseudowires (over MPLS or IP), VPLS instances, EVPN instance.

* The model is flexible in the scope of the VLAN Identifier space. I.e. by default VLAN Ids can be scoped locally to a single Ethernet-like trunk interface, but the scope is determined by the forwarding paradigm that is used.

A.2. IEEE 802.1Q Bridge Configuration Model Overview

The key features of the IEEE 802.1Q bridge configuration model can be summarized as:

* Each VLAN bridge component has a set of Ethernet interfaces that are members of that bridge. Sub-interfaces are not used, nor required in the 802.1Q bridge model.

* Within a VLAN bridge component, the VLAN tag in the packet is used, along with the destination MAC address, to determine how to forward the packet. Other forwarding paradigms are not supported by the 802.1Q model.

* Classification of traffic to a VLAN bridge component is based only on the Ethernet interface that it arrived on.

* VLAN Identifiers are scoped to a VLAN bridge component. Often devices only support a single bridge component and hence VLANs are scoped globally within the device.

* Feature configuration is specified in the context of the bridge, or particular VLANs on a bridge.

A.3. Possible Overlap Between the Two Models

Both models can be used for configuring similar basic layer 2 forwarding topologies. The 802.1Q bridge configuration model is optimised for configuring Virtual LANs that span across enterprises and data centers.

The sub-interface model can also be used for configuring equivalent Virtual LAN networks that span across enterprises and data centers, but often requires more configuration to be able to configure the equivalent constructs to the 802.1Q bridge model.

The sub-interface model really excels when implementing flexible L2 and L3 services, where those services may be handled on the same physical interface, and where the VLAN Identifier is being solely used to identify the customer or service that is being provided rather than a Virtual LAN. The sub-interface model provides more flexibility as to how traffic can be classified, how features can be applied to traffic streams, and how the traffic is to be forwarded.

Conversely, the 802.1Q bridge model can also be use to implement L2 services in some scenarios, but only if the forwarding paradigm being used to implement the service is the native Ethernet forwarding specified in 802.1Q - other forwarding paradigms such as pseudowires or VPLS are not supported. The 802.1Q bridge model does not implement L3 services at all, although this can be partly mitigated by using a virtual L3 interface construct that is a separate logical Ethernet-like interface which is a member of the bridge.

In conclusion, it is valid for both of these models to exist since they have different deployment scenarios for which they are optimized. Devices may choose which of the models (or both) to implement depending on what functionality the device is being designed for.

Authors' Addresses

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

Scott Mansfield (editor)
Ericsson
Email: scott.mansfield@ericsson.com

Network Working Group
Internet-Draft
Updates: 6020, 7950, 8407, 8525 (if approved)
Intended status: Standards Track
Expires: 2 September 2024

R. Wilton, Ed.
Cisco Systems, Inc.
R. Rahman, Ed.
Equinix
B. Lengyel, Ed.
Ericsson
J. Clarke
Cisco Systems, Inc.
J. Sterne
Nokia

1 March 2024

Updated YANG Module Revision Handling
draft-ietf-netmod-yang-module-versioning-11

Abstract

This document refines the RFC 7950 module update rules. It 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 a minimum revision suggestion to help document inter-module dependencies. It provides guidelines for managing the lifecycle of YANG modules and individual schema nodes. 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 2 September 2024.

Table of Contents

1. Introduction 3
1.1. Updates to YANG RFCs 4
2. Terminology and Conventions 4
3. Refinements to YANG revision handling 5
3.1. Updating a YANG module with a new revision 6
3.1.1. Backwards-compatible rules 7
3.1.2. Non-backwards-compatible changes 8
3.2. non-backwards-compatible extension statement 8
3.3. Removing revisions from the revision history 8
3.4. Examples for updating the YANG module revision history . 9
4. Guidance for revision selection on imports 12
4.1. Recommending a minimum revision for module imports . . . 13
4.1.1. Module import examples 14
5. New ietf-yang-status-conformance YANG module 15
5.1. Reporting how deprecated and obsolete nodes are handled 15
6. Guidelines for using the YANG module update rules 16
6.1. Guidelines for YANG module authors 16
6.1.1. Making non-backwards-compatible changes to a YANG module 17
6.2. Versioning Considerations for Clients 18
7. Module Versioning Extension YANG Modules 18
8. Security considerations 24
8.1. Security considerations for module revisions 24
8.2. Security considerations for the modules defined in this document 25
9. IANA Considerations 25
9.1. YANG Module Registrations 25
9.2. Guidance for versioning in IANA maintained YANG modules 26
10. References 27
10.1. Normative References 27
10.2. Informative References 28
Appendix A. Examples of changes that are NBC 30
Appendix B. Examples of applying the NBC change guidelines . . . 31
B.1. Removing a data node 31
B.2. Changing the type of a leaf node 31

B.3. Reducing the range of a leaf node 32
B.4. Changing the key of a list 32
B.5. Renaming a node 33
Contributors 33
Acknowledgments 34
Authors' Addresses 34

1. Introduction

The current YANG [RFC7950] module update rules require that updates of YANG modules preserve strict backwards compatibility. This causes 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 when importing YANG modules. Accepting that non-backwards-compatible changes do sometimes occur -- e.g., for bugfixes -- it is important to have mechanisms to report when these changes occur, and to manage their effect on clients and the broader YANG ecosystem.

Several other documents build on this document 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 defines a YANG extension that tags a YANG artifact with a version identifier 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 four 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.

* Updated guidance for revision selection on imports and a YANG extension statement allowing YANG module imports to document an earliest module revision that may satisfy the import dependency.

* Updates and augmentations to ietf-yang-library to report how "deprecated" and "obsolete" nodes are handled by a server.

* 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 4.1 describes a YANG extension statement to describe potential YANG import revision dependencies.

This document updates [RFC8407] section 4.7. Section 6 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 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.

However, using revision dates alone to identify revisions of a YANG module versioned with a branched revision history is likely to be confusing because the relationship between module revisions is no longer guaranteed to be chronologically ordered. Instead, for modules that may use a branched revision history, it is RECOMMENDED to use a version identifier, such as the one described in [I-D.ietf-netmod-yang-semver], that better describes the semantic relationship between the revisions.

For a given YANG module revision, revision B is defined as being derived from revision A, if revision A is listed in the revision history of revision B. Although this document allows for a branched revision history, a given YANG module revision history does not contain all revisions in all possible branches, it only lists those from which is was derived, i.e., the module revision's history describes a single path of derived revisions back to the root of the module's revision history.

A corollary to the text above is that the ancestry (derived relationship) between two module or submodule revisions cannot be determined by comparing the module or submodule revision date or version identifier alone - the revision history must be consulted.

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], could 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 backwards-compatible (BC) to the previous revision of the module. This document introduces a method to indicate that an non-backwards-compatible (NBC) change has occurred between module revisions: this is done by using a new "non-backwards-compatible" 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.

New module revisions SHOULD NOT contain NBC changes because they often create problems for clients, however they can be helpful in some scenarios, and hence are discouraged, but allowed. For example:

* Bugfixes, particularly where the likely client impact is low or the module is changed to reflect current server behavior.

* To mark nodes as obsolete (or remove them), after a suitable deprecation period.

* To refine new and unstable modules (or new and unstable nodes within existing, stable modules).

* Restructuring a module to add new functionality where the cost of adding the functionality in a BC manner is disproportionate to the expected benefits of greater client backwards compatibility.

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.

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 "recommended-min-date" substatements of an "import" statement, including adding new "revision-date" or "recommended-min-date" substatements, changing the argument of any "revision-date" or "recommended-min-date" substatetements, or removing any "revision-date" or "recommended-min-date" 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 extension statement

The "rev:non-backwards-compatible" 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" extension statement MUST be added as a substatement to the "revision" statement.

Adding, modifying or removing a "rev:non-backwards-compatible" 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 since it 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 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 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:

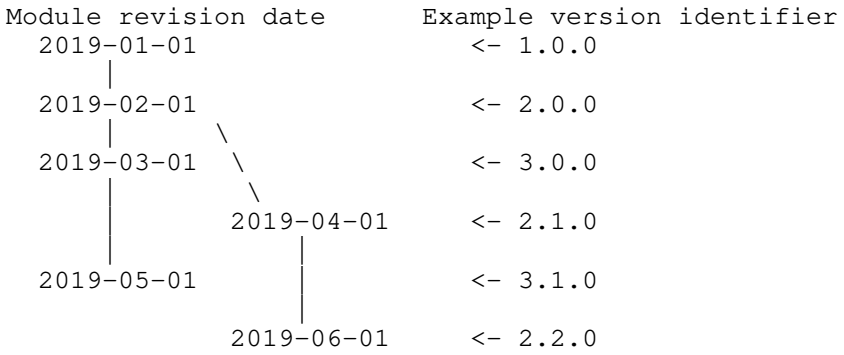
```
revision 2020-11-11 {
  rev:non-backwards-compatible;
}

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

revision 2020-06-07 {
}

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

revision 2019-10-21 {
}

revision 2019-03-04 {
}

revision 2019-01-02 {
}
```

In the revision history example above (with revision descriptions omitted for clarity), removing the revision history entry for 2020-02-10 would also remove the rev:non-backwards-compatible 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. Examples for updating the YANG module revision history

The following diagram, explanation, and module history illustrates how a branched revision history for a YANG module could be represented chronologically. To aid clarity, it makes use of both the "non-backwards-compatible" extension statement, and the "version" extension statement defined in [I-D.ietf-netmod-yang-semver]:

Example YANG module with branched revision history using version identifiers defined in [I-D.ietf-netmod-yang-semver].



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 2019-05-01:

```
module example-module {

  namespace "urn:example:module";
  prefix "prefix-name";

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

  description
    "to be completed";

  revision 2019-05-01 {
    ys:version 3.1.0;
    description "Add new functionality.";
  }

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

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

  revision 2019-01-01 {
    ys:version 1.0.0;
    description "Initial revision";
  }

  //YANG module definition starts here
}
```

Example module, revision 2019-06-01:

```
module example-module {

  namespace "urn:example:module";
  prefix "prefix-name";

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

  description
    "to be completed";

  revision 2019-06-01 {
    ys:version 2.2.0;
    description "Backwards-compatible bugfix to enhancement.";
  }

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

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

  revision 2019-01-01 {
    ys:version 1.0.0;
    description "Initial revision";
  }

  //YANG module definition starts here
}
```

4. Guidance for revision selection on imports

[RFC7950] and [RFC6020] allow YANG module "import" statements to optionally require the imported module to have a specific revision date. In practice, importing a module with an exact revision date can be too restrictive because it requires the importing module to be updated whenever any change to the imported module occurs, and hence section Section 6.1 suggests that authors do not restrict YANG module imports to exact revision dates.

Instead, for conformance purposes (section 5.6 of [RFC7950]), the recommended approach for defining the relationship between specific YANG module revisions is to specify the relationships outside of the

YANG modules, e.g., via YANG library [RFC8525], YANG packages [I-D.ietf-netmod-yang-packages], a filesystem directory containing a set of consistent YANG module revisions, or a revision control system commit label.

4.1. Recommending a minimum revision for module imports

Although the previous section indicates that the actual relationship constraints between different revisions of YANG modules should be specified outside of the modules, in some scenarios YANG modules are designed to be loosely coupled, and implementors may wish to select sets of YANG module revisions that are expected to work together. For these cases it can be helpful for a module author to provide guidance on a recommended minimum revision that is expected to satisfy an YANG import. E.g., the module author may know of a dependency on a type or grouping that has been introduced in a particular imported YANG module revision. Although there can be no guarantee that all derived future revisions from the particular imported module will necessarily also be compatible, older revisions of the particular imported module are very unlikely to ever be compatible.

This module introduces, for modules with a linear revision history that are versioned using revision dates, a new YANG extension statement to provide guidance to module implementors on a recommended minimum module revision of an imported module that is anticipated to be compatible. This statement has been designed to be machine-readable so that tools can parse the minimum revision extension statement and generate warnings if appropriate, but this extension statement does not alter YANG module conformance of valid YANG module versions in any way, and specifically it does not alter the behavior of the YANG module import statement from that specified in [RFC7950].

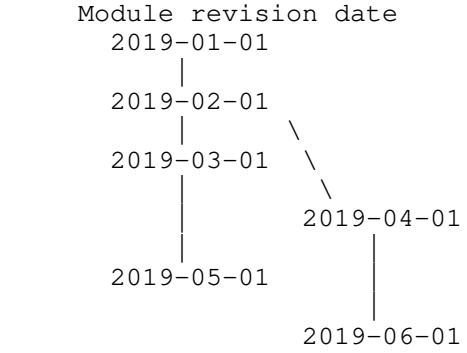
The ietf-revisions module defines the "recommended-min-date" extension statement, a substatement to the YANG "import" statement, to allow for a "minimum recommended date" to be documented:

> The argument to the "recommended-min-date" extension statement is a revision date.
>
> A particular revision of an imported module adheres to an import's "recommended-min-date" extension statement if the imported module's revision date is equal to or later than the revision date argument of the "recommended-min-date" extension statement in the importing module.
>
> Zero or one "recommended-min-date" extension statement is allowed for each parent "import" statement.

Adding, modifying or removing a "recommended-min-date" extension statement is a BC change.

4.1.1. Module import examples

Consider the example module "example-module" from Section 3.4 that is hypothetically available in the following revisions: 2019-01-01, 2019-02-01, 2019-03-01, 2019-04-01, 2019-05-01 and 2019-06-01. The relationship between the revisions is as before:



4.1.1.1. Example 1

This example recommends module revisions for import whose revision date is or comes after 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 the following revisions: 2019-02-01, 2019-03-01, 2019-04-01, 2019-05-01 and 2019-06-01.

```
import example-module {
  rev:recommended-min-date 2019-02-01;
}
```

4.1.1.2. Example 2

This example recommends module revisions for import whose revision date is or comes after 2019-04-01. It includes the following revisions: 2019-04-01, 2019-05-01 and 2019-06-01, even though revision 2019-05-01 may not contain what is desired from 2019-04-01. This shows that "recommended-min-date" is not well suited for a branched revision history, and is most helpful when a module is restricted to a linear chronological development history.

```
import example-module {
  rev:recommended-min-date 2019-04-01;
}
```

5. New ietf-yang-status-conformance YANG module

This document defines the YANG module, ietf-yang-status-conformance, that augments YANG library [RFC8525] with two leafs to indicate how a server implements deprecated and obsolete schema nodes.

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

```
module: ietf-yang-status-conformance
  augment /yanglib:yang-library/yanglib:schema:
    +--ro deprecated-nodes-implemented?   boolean
    +--ro obsolete-nodes-absent?          boolean
```

5.1. Reporting how deprecated and obsolete nodes are handled

The ietf-yang-status-conformance 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 by the server 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", which allows clients to determine the exact schema used by the server.

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" 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. 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.

6.1. Guidelines for YANG module authors

All IETF YANG modules MUST conform to this specification. In particular, sections: Section 3, Section 4, and the guidelines documented in this section.

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" statement MUST be added if there are NBC changes relative to the previous revision.

Removing old revision statements from a module’s revision history can cause a loss of visibility of when non-backwards-compatible changes were made, and hence it is RECOMMENDED to retain them. 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.

In cases where a revision dependency is helpful for a module import, the "rev:recommended-min-date" extension SHOULD be used 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

6.1.1. 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 accidentally 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.

6.2. 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.

7. Module Versioning Extension YANG Modules

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

```
<CODE BEGINS> file "ietf-yang-revisions@2024-02-19.yang"
module ietf-yang-revisions {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-revisions";
  prefix rev;

  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) 2024 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; 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 2024-02-19 {
       description
         "Initial version.";
       reference
         "XXXX: Updated YANG Module Revision Handling";
     }

     typedef revision-date {
       type string {
         pattern '[0-9]{4}-(1[0-2]|0[1-9])-(0[1-9]|[1-2][0-9]|3[0-1])';
       }
       description
         "A date associated with a YANG revision.

          Matches dates formatted as YYYY-MM-DD.";
       reference
         "RFC 7950: The YANG 1.1 Data Modeling Language";
     }

     extension non-backwards-compatible {
       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 recommended-min-date {
        argument revision-date;
        description
          "Recommends the revision of the module that may be imported to
           one whose revision date matches or is after the specified
           revision-date.

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

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

           Zero or one 'recommended-min-date' extension statement is
           allowed for each parent 'import' statement.

           A particular revision of an imported module adheres to an
           import's 'recommended-min-date' extension statement if the
           imported module's revision date is equal to or later than
           the revision date argument of the 'recommended-min-date'
           extension statement in the importing module.

           Adding, removing or updating a 'recommended-min-date'
           statement to an import is a backwards-compatible change.";
        reference
          "XXXX: Updated YANG Module Revision Handling; Section 4,
           Recommending a minimum revision for module imports";
      }
    }
    <CODE ENDS>
```

YANG module for status conformance

```
<CODE BEGINS> file "ietf-yang-status-conformance@2024-02-14.yang"
module ietf-yang-status-conformance {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-status-conformance";
  prefix ys-conf;

  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 provide an
     indication of how deprecated and obsolete nodes are handled by
     the server.

     Copyright (c) 2024 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; 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.

    revision 2024-02-14 {
      description
        "Initial revision";
      reference
        "XXXX: Updated YANG Module Revision Handling";
    }

    augment "/yanglib:yang-library/yanglib:schema" {
      description
        "Augmentations to the ietf-yang-library module to indicate how
         deprecated and obsoleted nodes are handled 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 5.1, 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 5.1, Reporting how deprecated and obsolete nodes
```

```
            are handled";
        }
      }
    }
    <CODE ENDS>
```

8. Security considerations

8.1. Security considerations for module revisions

As discussed in the introduction of this document, YANG modules occasionally undergo changes that are not backwards compatible. This occurs in both standards and vendor YANG modules despite the prohibitions in RFC 7950. RFC 7950 also allows nodes to change to status 'obsolete' which can change behavior and compatibility for a client.

The fact that YANG modules change in a non-backwards-compatible manner may have security implications. Such changes should be carefully considered, including the scenarios described below. The rev:non-backwards-compatible extension statement introduced in this document provides an alert that the module or submodule may contain changes that impact users and need to be examined more closely for both compatibility and potential security implications. Flagging the change reduces the risk of introducing silent exploitable vulnerabilities.

When a module undergoes a non-backwards-compatible change, a server may implement different semantics for a given leaf than a client using an older version of the module is expecting. If the particular leaf controls any security functions of the device, or is related to parts of the configuration or state that are sensitive from a security point of view, then the difference in behavior between the old and new revisions needs to be considered carefully. In particular, changes to the default of the leaf should be examined.

Implementors and users should also consider impact to data node access control rules (e.g. The Network Configuration Access Control Model (NACM) [RFC8341]) in the face of non-backwards-compatible changes. Access rules may need to be adjusted when a new module revision is introduced that contains a non-backwards-compatible change.

If the changes to a module or submodule have security implications, it is recommended to highlight those implications in the description of the revision statement.

8.2. Security considerations for the modules defined in this document

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

This document does not define any new protocol or data nodes that are writable.

This document updates YANG Library [RFC8525] with augmentations to include two boolean leafs that indicate whether status deprecated and status obsolete schema nodes are implemented by the server. These read-only augmentations do not add any new security considerations beyond those already present in [RFC8525].

9. IANA Considerations

9.1. 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-status-conformance
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-status-conformance module:

Name: ietf-yang-status-conformance

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-status-conformance

Prefix: ys-conf

Reference: [RFCXXXX]

9.2. 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" 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" 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.

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>.

[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>.

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[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>.

[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>.

[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>.

10.2. 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://datatracker.ietf.org/doc/html/draft-clacla-netmod-yang-model-update-06>.

[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://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-packages-03>.

[I-D.ietf-netmod-yang-schema-comparison]
Andersson, P. and R. Wilton, "YANG Schema Comparison", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-schema-comparison-02, 14 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-schema-comparison-02>.

[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-12, 2 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-12>.

[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://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-ver-selection-00>.

[I-D.ietf-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-versioning-reqs-09, 14 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-versioning-reqs-09>.

[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>.

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 6.1.1.

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.

Contributors

The following people made substantial contributions to this document:

Bo Wu
lana.wubo@huawei.com

Jan Lindblad
jlindbla@cisco.com

Acknowledgments

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

Benoit Claise
benoit.claise@huawei.com

Ebben Aries
exa@juniper.net

Juergen Schoenwaelder
j.shoenwaelder@jacobs-university.de

Mahesh Jethanandani
mjethanandani@gmail.com

Michael (Wangzitao)
wangzitao@huawei.com

Per Andersson
perander@cisco.com

Qin Wu
bill.wu@huawei.com

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.

Discussions 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, and Kent Watsen for their contributions and review comments.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Reshad Rahman (editor)
Equinix
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, 8525, 7950 (if approved)
Intended status: Standards Track
Expires: 5 September 2024

J. Clarke, Ed.
R. Wilton, Ed.
Cisco Systems, Inc.
R. Rahman
Equinix
B. Lengyel
Ericsson
J. Sterne
Nokia
B. Claise
Huawei

4 March 2024

YANG Semantic Versioning
draft-ietf-netmod-yang-semver-14

Abstract

This document specifies a YANG extension along with guidelines for applying an extended set of semantic versioning rules to revisions of YANG artifacts (e.g., modules and packages). Additionally, this document defines a YANG extension for controlling module imports based on these modified semantic versioning rules. This document updates RFCs 7950, 8407, and 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 5 September 2024.

Copyright Notice

Table of Contents

1. Introduction 3
2. Examples of How Versioning Is Applied To YANG Module Revisions 3
3. Terminology and Conventions 4
4. YANG Semantic Versioning 4
4.1. Relationship Between SemVer and YANG Semver 5
4.2. YANG Semantic Version Extension 5
4.3. YANG Semver Pattern 5
4.4. Semantic Versioning Scheme for YANG Artifacts 6
4.4.1. Branching Limitations with YANG Semver 8
4.4.2. YANG Semver with submodules 9
4.4.3. Examples for YANG semantic versions 9
4.5. YANG Semantic Version Update Rules 11
4.6. Examples of the YANG Semver Label 13
4.6.1. Example Module Using YANG Semver 13
4.6.2. Example of Package Using YANG Semver 14
5. Import Module by YANG Semantic Version 15
5.1. The recommended-min-version Extension 15
5.2. Import by YANG Semantic Version Rules 16
6. Guidelines for Using Semver During Module Development 16
6.1. Pre-release Version Precedence 18
6.2. YANG Semver in IETF Modules 18
6.2.1. Guidelines for IETF Module Development 18
6.2.2. Guidelines for Published IETF Modules 19
7. Updates to ietf-yang-library 19
7.1. YANG library versioning augmentations 19
7.1.1. Advertising version 20
8. YANG Modules 20
9. Contributors 26
10. Acknowledgments 27
11. Security Considerations 27
12. IANA Considerations 27
12.1. YANG Module Registrations 28
12.2. Guidance for YANG Semver in IANA maintained YANG modules and submodules 29
13. References 29
13.1. Normative References 29

13.2. Informative References 30
Appendix A. Example IETF Module Development 32
Authors' Addresses 33

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.

This document defines a YANG extension that tags a YANG artifact (i.e., YANG modules, YANG submodules, and YANG packages [I-D.ietf-netmod-yang-packages]) with a version identifier that adheres to extended semantic versioning rules [SemVer]. The goal being to add a human readable version identifier that provides compatibility information for the YANG artifact without needing to compare or parse its body. The version identifier and rules defined herein represent the RECOMMENDED approach to apply versioning to IETF YANG artifacts. This document defines augmentations to ietf-yang-library to reflect the version of YANG modules within the module-set data.

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. Examples of How Versioning Is Applied To YANG Module Revisions

The following diagram illustrates how the branched revision history and the YANG Semver version extension statement could be used:

Example YANG module with branched revision history.

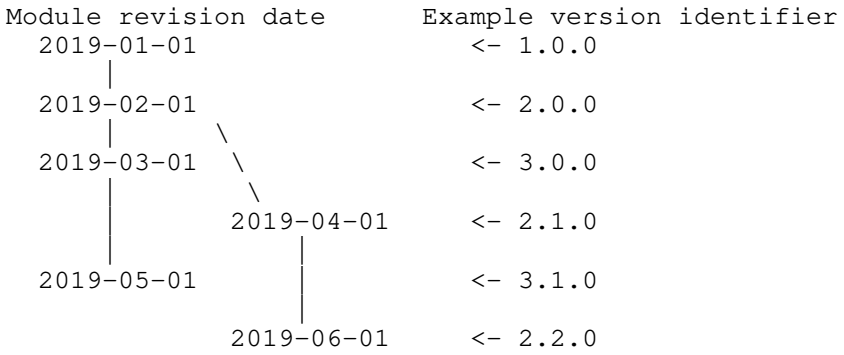


Figure 1

The tree diagram above illustrates how an example module's revision history might evolve, over time.

3. 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 version identifier that is consistent with the extended set of semantic versioning rules, based on [SemVer], defined within this document.

4. 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.

4.1. 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 allows 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.

4.2. YANG Semantic Version Extension

The ietf-yang-semver module defines a "version" extension -- a substatement to a module or submodule's "revision" statement -- that takes a YANG semantic version as its argument and specified the version for the given module or submodule. The syntax for the YANG semantic version is defined in a typedef in the same module and described below.

4.3. YANG Semver Pattern

YANG artifacts that employ semantic versioning as defined in this document MUST use a version identifier 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 6. 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.

The ietf-yang-semver module included in this document defines an extension to apply a YANG Semver identifier to a YANG artifact as well as a typedef that formally specifies the syntax of the YANG Semver.

4.4. Semantic Versioning Scheme for YANG Artifacts

This document defines the YANG semantic versioning scheme that is used for YANG artifacts. The versioning identifier 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 lifecycles 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 use 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 identifiers 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 'ys:version' 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 identifier. 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

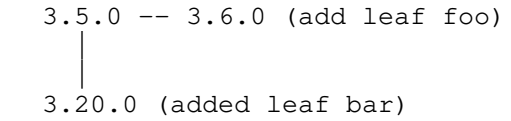
The '_COMPAT' modifier string is "sticky". Once a revision of a module has a modifier in the version identifier, then all subsequent modules in that branch (i.e., those with the same X.Y version digits) will also have a modifier. The modifier can change from "_compatible" to "_non_compatible" in a subsequent version, 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 versions do not give the false impression of compatibility between two potentially non-compatible versions. If "_non_compatible" was removed, for example between versions "3.3.2_non_compatible" and "3.3.3" (where "3.3.3" was simply an editorial change), then comparing versions "3.3.3" to "3.0.0" would look like they are backwards compatible when they are not (since "3.3.2_non_compatible" was on the same MAJOR.MINOR branch 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.

4.4.1. Branching Limitations with YANG Semver

YANG artifacts that use the YANG Semver version 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 branching approach using the following YANG Semver identifiers is not supported:



In this case, given only the YANG Semver identifiers 3.6.0 and 3.20.0, 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 branching, where two versions on the same branch have different backwards compatible changes is allowed in [I-D.ietf-netmod-yang-module-versioning].

4.4.2. 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 4.3 and Section 4.4 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 resulting 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

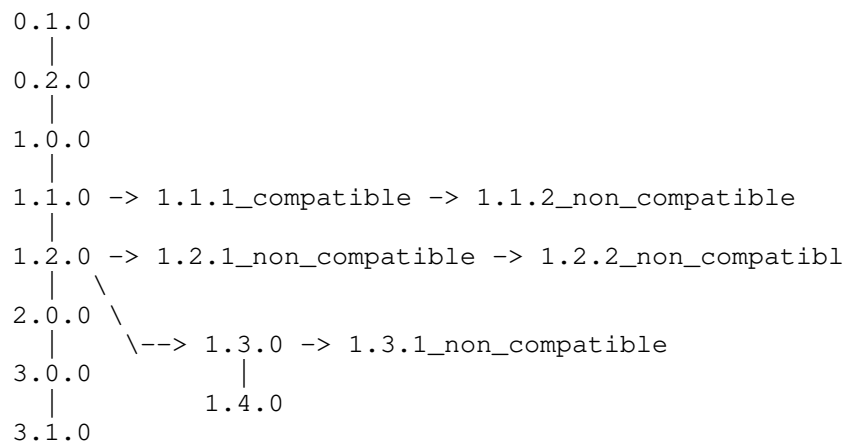
3. The submodule with the schema definitions added has backwards-compatible changes

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).

4.4.3. 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 branching relationships between the versions. It does not describe the chronology of the versions (i.e. when in time each version was published relative to the other versions).

The following description lists an example of what the chronological order of the versions could look like, from oldest version 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)

4.5. YANG Semantic Version Update Rules

When a new version of an artifact is produced, then the following rules define how the YANG semantic version for the new artifact is calculated, based on the changes between the two artifact versions, and the YANG semantic version of the original artifact 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 identifier depends on the format of the current version identifier:

   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 identifiers 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 6 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 6.

Although artifacts SHOULD be updated according to the rules above, which specify the recommended (and minimum required) update to the version identifier, the following rules MAY be applied when choosing a new version identifier:

1. An artifact author MAY update the version identifier 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 versions. That is, an artifact's version history could be 1.0.0, 1.1.0, and 1.3.0 where 1.2.0 is skipped.

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.

4.6. Examples of the YANG Semver Label

4.6.1. Example Module Using YANG Semver

Below is a sample YANG module that uses YANG Semver based on the rules defined in this document.

```
module example-versioned-module {
  yang-version 1.1;
  namespace "urn:example:versioned:module";
  prefix "exvermod";

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

  description
    "to be completed";

  revision 2017-08-30 {
    description "Backport 'wibble' leaf";
    ys:version 1.2.2_non_compatible;
  }

  revision 2017-07-30 {
    description "Rename 'baz' to 'bar'";
    ys:version 1.2.1_non_compatible;
    rev:non-backwards-compatible;
  }

  revision 2017-04-20 {
    description "Add new functionality, leaf 'baz'";
    ys:version 1.2.0;
  }

  revision 2017-04-03 {
    description "Add new functionality, leaf 'foo'";
    ys:version 1.1.0;
  }
```

```
      revision 2017-02-07 {
        description "First release version.";
        ys:version 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";
      //   ys:version 0.2.0;
      //   rev:non-backwards-compatible; // optional
      //                          // (theoretically no
      //                          // 'previous released version')
      // }

      // revision 2017-01-26 {
      //   description "Initial module version";
      //   ys:version 0.1.0;
      // }

      //YANG module definition starts here
    }
```

4.6.2. Example of Package Using YANG Semver

Below is an example YANG package that uses the YANG Semver version identifier based on the rules defined in this document. Note: '\' line wrapping per [RFC8792].

```
{
  "ietf-yang-instance-data:instance-data-set": {
    "name": "example-yang-pkg",
    "content-schema": {
      "module": "ietf-yang-packages@2022-03-04"
    },
    "timestamp": "2022-12-06T17:00:38Z",
    "description":  ["Example of a Package  \
       using YANG Semver"],
    "content-data": {
      "ietf-yang-packages:packages": {
        "package": [
          {
            "name": "example-yang-pkg",
            "version": "1.3.1",
            ...
          }
        ]
      }
    }
  }
}
```

Figure 2

5. Import Module by YANG Semantic Version

[I-D.ietf-netmod-yang-module-versioning] allows for imports to be done based on the earliest supported date and later using the rev:recommended-min-date extension. This section defines a similar extension for controlling import by YANG semantic version, as well as the rules for how imports are resolved.

5.1. The recommended-min-version Extension

The ietf-yang-semver module defines a "recommended-min-version" extension -- a substatement to the "import" statement -- that takes a YANG semantic version as its argument and specifies that the minimum version of the associated module being imported SHOULD be greater than or equal to the specified value. The specific conditions for determining if a module's version is greater than or equal is defined in Section 5.2 below. Multiple recommended-min-version statements MAY be specified. If there are multiple recommended-min-version statements, they are treated as a logical OR. Removing recommended-min-version statements is considered a backwards compatible change. An example use is:

```
import example-module {
    ys:recommended-min-version 3.0.0;
}
```

5.2. Import by YANG Semantic Version Rules

A module to be imported is considered viable with recommended-min-version if it meets one of the following conditions:

1. Has the same MAJOR and MINOR version numbers and same or greater PATCH number.

2. Has the same MAJOR version number and greater MINOR number. In this case the PATCH number is ignored.

3. Has a greater MAJOR version number. In this case MINOR and PATCH numbers are ignored.

In all cases, the "_compatible" and "_non_compatible" version modifiers are ignored.

If the recommended-min-version is specified as 3.1.0, the following examples would be viable:

3.1.1 (by condition 1 above)

3.2.0 (by condition 2 above)

4.1.2 (by condition 3 above)

3.1.1_compatible (by condition 1 above, noting that modifiers are ignored)

3.1.2_non_compatible (by condition 1 above, noting that modifiers are ignored)

If an import by recommended-min-version cannot locate a module with a version that is viable according to the conditions above, the YANG compiler SHOULD emit a warning, and then continue to resolve the import based on established [RFC7950] rules.

6. 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 the YANG Semver versioning 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 number. For example, an initial module or submodule version might be either 0.0.1 or 1.0.0-alpha.1. If the authors choose to use the 0 MAJOR version number scheme, they MAY switch to the pre-release scheme with a MAJOR version number of 1 when the module or submodule is nearing initial release (e.g., a module's or submodule's version 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 versioning 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 version 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 identifier (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 version 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).

6.1. Pre-release Version Precedence

As a module or submodule is developed, the scope of the work may change. That is, while a released module or submodule with version 1.0.0 is initially intended to become 2.0.0 in its next released 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 identifiers 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.

6.2. YANG Semver in IETF Modules

All published IETF modules and submodules MUST use YANG semantic versions in their revisions.

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 version MUST use the target (i.e., intended) MAJOR and MINOR version components with a 0 PATCH version number. If the intended RFC release will be non-backwards-compatible with the current RFC release, the MINOR version number MUST be 0.

6.2.1. Guidelines for IETF Module Development

All IETF modules and submodules in development MUST use the whole document name as a pre-release version identifier, 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 versions 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.

6.2.2. Guidelines for Published IETF Modules

For IETF YANG modules and submodules that have already been published, versions 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 4.5. 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).

7. 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-semver, that augments YANG library [RFC8525] with a version leaf for modules and submodules.

7.1. YANG library versioning augmentations

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

```
module: ietf-yang-library-semver

  augment /yanglib:yang-library/yanglib:module-set/yanglib:module:
    +--ro version?   ys:version
  augment /yanglib:yang-library/yanglib:module-set/yanglib:module
            /yanglib:submodule:
    +--ro version?   ys:version
  augment /yanglib:yang-library/yanglib:module-set
            /yanglib:import-only-module:
    +--ro version?   ys:version
  augment /yanglib:yang-library/yanglib:module-set
            /yanglib:import-only-module/yanglib:submodule:
    +--ro version?   ys:version
```

Figure 3

7.1.1. Advertising version

The ietf-yang-library-semver YANG module augments the "module" and "submodule" lists in ietf-yang-library with "version" leafs to optionally declare the version identifier associated with each module and submodule.

8. YANG Modules

This YANG module contains the typedef for the YANG semantic version and the identity to signal its use.

```
<CODE BEGINS> file "ietf-yang-semver@2024-03-01.yang"
module ietf-yang-semver {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-semver";
  prefix ys;

  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) 2024 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).

       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.

       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 ys:version to "1.0.0".

    revision 2024-03-01 {
      ys:version "1.0.0-draft-ietf-netmod-yang-semver-13";
      description
        "Initial revision";
      reference
        "RFC XXXX: YANG Semantic Versioning.";
    }

    /*
     * Extensions
     */

    extension version {
      argument yang-semantic-version;
      description
```

```
        "The version extension can be used to provide an additional
         identifier associated with a module or submodule
         revision.

         The format of the version extension argument MUST conform
         to the 'version' typedef defined in this module.

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

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

         Adding a version is a backwards-compatible
         change.  Changing or removing an existing version in
         the revision history is a non-backwards-compatible
         change, because it could impact any references to that
         version.";
      reference
        "XXXX: YANG Semantic Versioning;
         Section 3.2, YANG Semantic Version Extension";
    }

    extension recommended-min-version {
      argument yang-semantic-version;
      description
        "Recommends the versions of the module that may be imported to
         one that is greater than or equal to the specified version.

         The format of the recommended-min-version extension argument
         MUST conform to the 'version' typedef defined in this module.

         The statement MUST only be a substatement of the import
         statement.  Zero, one or more 'recommended-min-version'
         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 'recommended-min-version'
         statements is an acceptable recommended version for
         import.

         A particular version of an imported module adheres to an
         import's 'recommended-min-version' extension statement if one
         of the following conditions are met:
```

```
          * Has the same MAJOR and MINOR version numbers and same or
            greater PATCH number.
          * Has the same MAJOR version number and greater MINOR number.
            In this case the PATCH number is ignored.
          * Has a greater MAJOR version number.  In this case
            MINOR and PATCH numbers are ignored.

          Adding, removing or updating a 'recommended-min-version'
          statement to an import is a backwards-compatible change.";
        reference
          "XXXX: YANG Semantic Versioning; Section 4,
           Import Module by Semantic Version";
      }

      /*
       * Typedefs
       */

      typedef version {
        type string {
          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 version identifier are defined in the
           reference for this typedef.";
        reference
          "RFC XXXX: YANG Semantic Versioning.";
      }
    }
    <CODE ENDS>

    This YANG module contains the augmentations to the ietf-yang-library.

    <CODE BEGINS> file "ietf-yang-library-semver@2024-03-02.yang"
    module ietf-yang-library-semver {
      yang-version 1.1;
      namespace
        "urn:ietf:params:xml:ns:yang:ietf-yang-library-semver";
      prefix yl-semver;

      import ietf-yang-semver {
        prefix ys;
        reference
          "XXXX: YANG Semantic Versioning";
      }
      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
       and submodule level version identifiers.

       Copyright (c) 2024 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.

       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 ys:version with 1.0.0 and
     // remove this note.

     revision 2024-03-02 {
       ys:version "1.0.0-draft-ietf-netmod-yang-semver-14";
       description
         "Initial revision";
       reference
         "XXXX: YANG Semantic Versioning";
     }

     // library 1.0 modules-state is not augmented with version

     augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
       description
         "Add a version to module information";
       leaf version {
         type ys:version;
         description
           "The version associated with this module revision.
            The value MUST match the version value in the
            specific revision of the module loaded in this module-set.";
         reference
           "XXXX: YANG Semantic Versioning;
            Section 7.1.1, Advertising version";
       }
     }

     augment
       "/yanglib:yang-library/yanglib:module-set/yanglib:module/"
     + "yanglib:submodule" {
       description
         "Add a version to submodule information";
       leaf version {
         type ys:version;
         description
           "The version associated with this submodule revision.
            The value MUST match the version value in the
            specific revision of the submodule included by the module
            loaded in this module-set.";
         reference
           "XXXX: YANG Semantic Versioning;
            Section 7.1.1, Advertising version";
       }
```

```
    }

    augment "/yanglib:yang-library/yanglib:module-set/"
          + "yanglib:import-only-module" {
      description
        "Add a version to module information";
      leaf version {
        type ys:version;
        description
          "The version associated with this module revision.
           The value MUST match the version value in the
           specific revision of the module included in this
           module-set.";
        reference
          "XXXX: YANG Semantic Versioning;
           Section 7.1.1, Advertising version";
      }
    }

    augment "/yanglib:yang-library/yanglib:module-set/"
          + "yanglib:import-only-module/yanglib:submodule" {
      description
        "Add a version to submodule information";
      leaf version {
        type ys:version;
        description
          "The version associated with this submodule revision.
           The value MUST match the version 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 7.1.1, Advertising version";
      }
    }
  }
  <CODE ENDS>
```

9. Contributors

The following people made substantial contributions to this document:

Bo Wu
lana.wubo@huawei.com

Jan Lindblad
jlindbla@cisco.com

Figure 4

10. Acknowledgments

This document grew out of the YANG module versioning design team that started after IETF 101. The 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), Per Andersson, Qin Wu, Reshad Rahman, Tom Hill, 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.

We would also like to thank Joseph Donahue from the SemVer.org project for his input on SemVer use and overall document readability.

11. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

That said, the YANG module in this document does not define any writeable nodes. The extensions defined are only used to document YANG artifacts.

12. IANA Considerations

12.1. YANG Module Registrations

This document requests IANA to register URIs 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-semver

Registrant Contact: The IESG.

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

URI: urn:ietf:params:xml:ns:yang:ietf-yang-library-semver

Registrant Contact: The IESG.

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

The following YANG modules are 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: ys

Reference: [RFCXXXX]

The ietf-yang-library-semver module:

Name: ietf-yang-library-semver

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

Prefix: yl-semver

Reference: [RFCXXXX]

12.2. 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 version identifier for all new revisions, as defined in Section 4.

The YANG Semver version associated with the new revision MUST follow the rules defined in Section 4.5.

Note: For IANA maintained YANG modules and submodules that have already been published, versions 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 4.5.

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.

13. References

13.1. 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>.

[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>.

[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>.

[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-11, 1 March 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-netmod-yang-module-versioning/>.

13.2. 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://datatracker.ietf.org/doc/html/draft-clacla-netmod-yang-model-update-06>.

[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://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-packages-03>.

[I-D.ietf-netmod-yang-schema-comparison]
Andersson, P. and R. Wilton, "YANG Schema Comparison", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-schema-comparison-02, 14 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-schema-comparison-02>.

[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>.

[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>.

[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>.

[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>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

[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>.

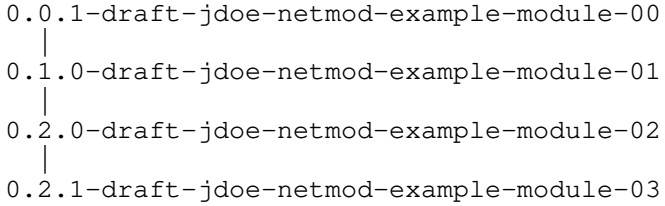
[IfTypeYang] "iana-if-type YANG Module", <https://www.iana.org/assignments/iana-if-type/iana-if-type.xhtml>.

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

Appendix A. Example IETF Module Development

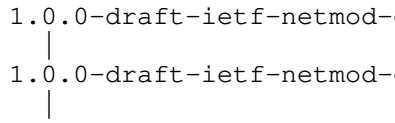
Assume a new YANG module is being developed in the netmod working group in the IETF. Initially, this module is being developed in an individual internet draft, draft-jdoe-netmod-example-module. The following represents the initial version tree (i.e., value of ys:version) of the module as it's being initially 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.

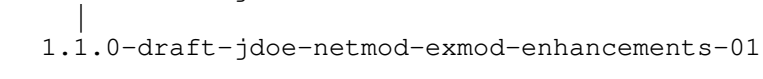
Continued version progression after adoption:



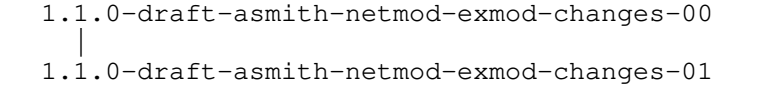
At this point, the draft is standardized and becomes RFC12345 and the YANG module version becomes 1.0.0.

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.

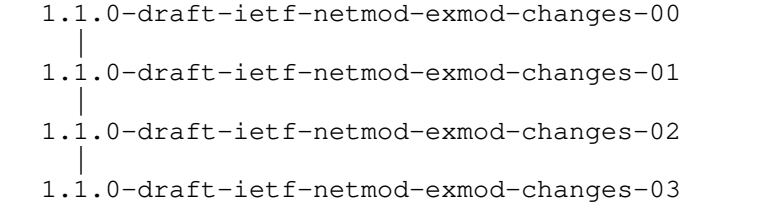
Parallel development for next module revision (track 1):



In parallel with (track 2):



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 progression continues.



The draft is standardized, 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
Equinix
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: 5 September 2024

J. Quilbeuf
B. Claise
T. Joubert
Huawei

4 March 2024

YANG Full Embed
draft-jouqui-netmod-yang-full-include-01

Abstract

YANG lacks re-usability of models defined outside of the grouping and augmentation mechanisms. For instance, it is almost impossible to reuse a model defined for a device in the context of the network, i.e by encapsulating it in a list indexed by device IDs. [RFC8528] defines the YANG mount mechanism, partially solving the problem by allowing to mount an arbitrary set of schemas at an arbitrary point. However, YANG mount is only focusing on deploy or runtime. This document aims to provide the same mechanism at design time.

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 5 September 2024.

Table of Contents

1. Introduction 2
2. Terminology 4
3. Full Embed 4
3.1. Definition 5
3.2. Limitations 5
3.3. Allowed sub-statements 6
4. ietf-full-embed YANG module 6
5. Security Considerations 8
6. IANA Considerations 8
7. Contributors 8
8. Open issues 8
8.1. Parent-nodes mechanism from schema mount 8
9. References 9
9.1. Normative References 9
9.2. Informative References 10
Appendix A. Changes between revisions 11
Appendix B. Examples 11
B.1. Example using YANG Full Embed 12
B.2. Using YANG Schema Mount 13
B.3. Support Files 14
Acknowledgements 17
Authors' Addresses 17

1. Introduction

[RFC8528] introduces the challenges of reusing existing YANG modules, especially when including the full subtree of YANG module under a specific node of another module. In that RFC, three different phases of data model life cycle are identified: "design time", "implementation time" and "run time". Only the last two are covered. We focus here on the first phase of the life cycle, that is inserting modules at design time.

We identified some use cases that require this design time definition of which modules need to be included in the top-level module. The have in common the need to re-use YANG modules defined for the devices in the context of a network-level module. Also, they both aim to define a model that is independent of the underlying devices.

* The use case that triggered the creation of this document is [I-D.ietf-opsawg-collected-data-manifest]. In this draft, the goal is to provide a YANG model giving the context in which YANG-

push [RFC8641] data are collected so that they can be exploited a posteriori. To get the full context, we need the hardware and os version of each device, but also the list of YANG modules supported by the devices and the parameters for the YANG-push subscriptions. For the last two items YANG Library [RFC8525] and YANG Push [RFC8641] provide good and standard modules for representing this information at the device level. However, the data manifests need to be considered at the network level, so that we can distinguish between the devices from which they come. In YANG, that means including them in a list indexed by the device id, which proves out to be difficult without copy-pasting the original modules.

* A similar use case is the digital map [I-D.havel-opsawg-digital-map], where the goal is to build a model of the network. In particular, to model the devices a lot of standard modules have already been defined by the IETF and there is a need to reuse these modules to build this larger network model. The IVY workgroup (https://datatracker.ietf.org/wg/ivy/about/) might also rely on the pattern of re-using device level modules into a network model.

YANG Schema Mount [RFC8528] and Peer Mount [I-D.clemm-netmod-peermount] focus on mounting a given part of a an existing data instance into another data instance. Although the final goal is the same: being able to reuse modules defined elsewhere in order to avoid redefining them, the approach is more focused on the runtime than the design time. In the first case, the mapping between the mount points and the existing modules to be mounted at that mount point is left to the NETCONF [RFC6241] server. Thus, to guarantee that the contents under a given mount point conforms to a predefined schema requires the proper configuration of the server. In the case of Peer mount, the focus is on synchronizing a given subtree of a server (remote or local) with a subtree of the local server. Again, the contents under the local subtree cannot be enforced from the design time.

The notion of reusing an existing schema within a new schema is not new. Several schema definition languages propose this feature, such as RELAX NG (https://books.xmlschemata.org/relaxng/relax-CHP-10-SECT-1.html), Protobuf (https://protobuf.dev/programming-guides/proto3/#other/) or json-schema (https://json-schema.org/understanding-json-schema/structuring#dollarref).

In this document, we propose a new extension, named full embed. This extension enables reusing imported modules by rooting them at an arbitrary point of the data model. The concept of mount point from [RFC8528] is replaced by an anydata statement containing list of

"full:embed" statement, each statement corresponding to the inclusion of one imported module at that location. In that sense, the design time solution is a pure YANG solution that does not rely on external configuration to specify the list of mounted modules, hence the term full embed rather than mount. Also, we use 'embed' not to conflict with the native 'include' statement in YANG [RFC7950].

The obtained data model that we want to associate to our construct is similar to the one obtained by specifying a mount point and binding it to the same set of modules. Therefore, we can reuse the concepts of the YANG schema mount to define the semantics of our new extension.

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 following terms are defined in [RFC7950]:

* data model

* data node

The following terms are defined in [RFC8528]:

* mount point

3. Full Embed

The full embed mechanism defined in this document completes [RFC8528], by providing a mechanism to "mount" modules at design time. Supporting mounting modules at this step of the data model life cycle is left out of scope in [RFC8528].

The approach for supporting the full embed mechanism is to keep the semantics of [RFC8528] for the resulting data model. In [RFC8528], the list of modules to mount in each mount point is left to the NETCONF server. In this document, we propose the full embed mechanism to define this mapping directly in the YANG module that embeds the mounted modules.

To ensure interoperability with clients that do not support the full embed extension, the full embed statement can only appear within an anydata node. Clients that do not support the extension will see the

contents of the embedded model as arbitrary data. Clients that support the extension will be able to interpret the contents of the anydata node according to the semantics of the embedded YANG modules.

In the sequel, we use "full" as the prefix for the module 'ietf-yang-full-embed' (see Section 4). Thus "full:embed" refers to the extension 'embed' defined in that module.

3.1. Definition

The "full:embed" statement can appear as a sub-statement of anydata.

The "full:embed" statement takes as an argument a prefix, that must be the prefix associated to an imported module. Modules can contain multiple uses of the "full:embed" statement. An "anydata" statement MAY contain multiple uses of the "full:embed" statement. These multiple uses define the full list of modules to be embedded, rooted in the anydata node where the "full:embed" statement is used.

The "full:embed" statement can be interpreted using YANG Schema Mount [RFC8528], by following these steps:

1. For each anydata node containing a set of "full:embed" statement, replace them by a container containing single "mount-point" with a unique label.

2. Declare each of these "mount-points" as "shared-schema" in the data model defined in [RFC8528].

3. In the instance corresponding to each "mount-point", define the ietf-yang-library [RFC8525] to include a module-set (at '/yang-library/module-set/) with the following. The list 'module' contains an entry for every module referred to in the set of "full:embed" statements corresponding to the "mount-point". Additionally, the list 'module' contains an entry for "ietf-yang-library" as it is needed by YANG Schema mount. As usual, the list 'imported-modules' contains the list of dependencies needed by the modules in the 'module' list.

An example of module using "full:embed" and its translation into a similar YANG Schema mount version is presented in Appendix B.

3.2. Limitations

A module MUST NOT use the "full:embed" statement with its own prefix as argument. This rule prevents any infinite recursion in the mounted schemas.

As for YANG Schema Mount, the set of embedded modules is an independent YANG context, where every reference (for instance leaf-ref, augment, when) is contained in that context. It is not possible for an embedded module to refer to the embedding module, which would be rejected by the compiler anyway because it would create a dependency loop. If a server supports a module both at top-level and embedded in another module, the corresponding data instances are disjoint.

Activation of the features for the embedded module follows the same rules as for normal module. Therefore its not possible to activate some features for some embedded modules only. The feature is either supported by the server and then activated for every module (embedded or not) or not supported and then deactivated for all modules (embedded or not).

3.3. Allowed sub-statements

The following sub-statements are allowed in the "full:embed" statement:

* when

* if-feature

Both statements have the same meaning as in [RFC7950]. The when statement MUST NOT refer to nodes which are in the embedded module designated by the "full:embed" statement.

4. ietf-full-embed YANG module

We present in this section the YANG module defining the "full-embed" extension. The module in itself defines solely the 'embed' extension. A module importing this extension SHOULD use the prefix 'full', so that the statement reads "full:embed" when used in the code.

```
<CODE BEGINS> file "ietf-full-embed@2023-11-03.yang"
module ietf-yang-full-embed {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-full-embed";
  prefix full;

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

```
      Editor:   ";
    description
      "This module defines a YANG extension statement that can be used
       to incorporate data models defined in other YANG modules in a
       module.

       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.

       Copyright (c) 2023 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;
       see the RFC itself for full legal notices.";

    revision 2023-11-05 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: YANG Full Embed";
    }

    extension embed {
      argument prefix;
      description
        "The argument 'prefix' MUST be the prefix of a module imported
         by the calling module.

         The 'embed' statement MUST NOT be used in a YANG version 1
         module, neither explicitly nor via a 'uses' statement.

         The 'embed' statement MAY be present as a substatement of
         'anydata' and MUST NOT be present elsewhere.

         Whenever a sequence of 'embed' statements is used, the schema
         tree defined by the set of the included modules is inserted
         in the schema tree of the calling module, at the place where
         the sequence is declared";
```

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

5. Security Considerations

TODO

6. IANA Considerations

TODO

7. Contributors

8. Open issues

* What name should we give to this draft? Any suggestions instead of full embed?

* Do we want to support the parent-nodes mechanism from [RFC8528]? (see below)

* Do we allow full embed into an augment? We could even relax no self-reference to have a module embed itself into another by augmenting it?

* Does this mechanism already exist?

* Do we want to add a partial embed with an xpath instead of just the prefix? The goal would be to include only part of a module. This complexifies a bit the validation as leaf-ref, must, when and other statement involving Xpath will need to be reinterpreted in that new context.

8.1. Parent-nodes mechanism from schema mount

YANG Schema Mount includes a mechanism to make some nodes from the embedding model available to the embedded model for validation purposes. We could achieve the same by adding a second extension, which can also only appear under a "full:embed" nodes. That extension, for instance named "full:embed-parent-refs" would take a Xpath expression as the in the "parent-reference" leaflist defined in the YANG Schema Mount and would have the same semantics. If several XPath are needed for clarity, the statement can be repeated with several values.

As an example, Figure 1 restates the parent-references example from [RFC8528] using this new extension. We might want to put some restrictions on the nodes that can be referred to in the Xpath argument.

```
...
import "ietf-routing" {
  prefix "rt";
}
import "ietf-interfaces" {
  prefix "if";
}

...
container network-instances {
  list network-instance {
    leaf name {...}
    anydata root {
      full:embed "rt" {
        full:embed-parent-refs "if:interfaces/if:interface[\
              ni:bind-network-instance-name = current()/../ni:name]";
      }
      // other full:embed if needed
    }
  }
}
```

Figure 1: Pseudo-YANG example of parent-references from [RFC8528] with "full:embed"

9. References

9.1. Normative References

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

[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>.

[RFC8528] Bjorklund, M. and L. Lhotka, "YANG Schema Mount", RFC 8528, DOI 10.17487/RFC8528, March 2019, <https://www.rfc-editor.org/info/rfc8528>.

9.2. Informative References

[I-D.clemm-netmod-peermount]
Clemm, A., Voit, E., Guo, A., and I. D. Martinez-Casanueva, "Mounting YANG-Defined Information from Remote Datastores", Work in Progress, Internet-Draft, draft-clemm-netmod-peermount-02, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-clemm-netmod-peermount-02>.

[I-D.havel-opsawg-digital-map]
Havel, O., Claise, B., de Dios, O. G., Elhassany, A., Graf, T., and M. Boucadair, "Modeling the Digital Map based on RFC 8345: Sharing Experience and Perspectives", Work in Progress, Internet-Draft, draft-havel-opsawg-digital-map-01, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-havel-opsawg-digital-map-01>.

[I-D.ietf-opsawg-collected-data-manifest]
Claise, B., Quilbeuf, J., Lopez, D. R., Dominguez, I., and T. Graf, "A Data Manifest for Contextualized Telemetry Data", Work in Progress, Internet-Draft, draft-ietf-opsawg-collected-data-manifest-03, 4 March 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-opsawg-collected-data-manifest/>.

[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>.

[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>.

[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>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

Appendix A. Changes between revisions

00 -> 01

* Renamed full include -> full embed

* Require extension to appear in anydata for clients not supporting extension

* Allow "if-feature" and "must" as sub-statement of full:embed, explain that feature work at server level

Appendix B. Examples

In this section we present some minimalistic examples in order to illustrate the "full:embed" statement. For these examples, we are in a situation where we have a device-level module already defined and we want to have a network-level module that represent a list of device, each having an independent instance of the device-level module. This situation might arise if we want to simplify the network management by presenting a unified model for the network. In that case, the heterogeneity of the devices should be handled by mapping their model to the device-level module (which is clearly out of scope for this draft).

In our simplistic example, the device-level module simply exposes the hostname and the cpu-usage of the device. Note that we cannot modify this device-level module, because in a more realistic example we would be reusing standard modules. The tree representation ([RFC8340]) of the 'device-level' module is depicted in Figure 2.

```
module: device-level
  +--rw hostname     string
  +--ro cpu-usage?   int8
```

Figure 2: YANG Tree representation of the device-level module.

For the network-level module, we have a list of devices indexed by their 'device-id'. The tree representation ([RFC8340]) of such a module is depicted in Figure 3.

```
module: network-level-stub
  +--rw devices
     +--rw device* [device-id]
        +--rw device-id    string
```

Figure 3: YANG Tree representation of a stub for the network-level module

The goal is now to complete this stub so that the full contents of the 'device-level' is added under the "device" list.

B.1. Example using YANG Full Embed

We propose in this section a YANG module for 'network-level'. The YANG code is presented in Figure 4.

```
module network-level {
  yang-version 1.1;
  namespace "urn:network-level";
  prefix "net-l";

  import "ietf-yang-full-include" {
    prefix "full";
  }

  import "device-level" {
    prefix "dev-l";
  }

  container devices {
    list device {
      key device-id;
      leaf device-id {
        type string;
      }
      anydata device-content {
           full:include "dev-l";
      }
    }
  }
}
```

Figure 4: Version of the network-level module using full:embed

At the moment, this code is accepted by the YANG compilers, but since the extension is not implemented, it simply ignores it. Note that all the information (which modules to embed, where to embed them) is defined in this module. More specifically, the line 'full:embed "dev-l";' states that the full schema of the 'device-level' module, identified by its prefix "dev-l" must be embedded at that location. By adding more occurrences of "full:embed" there, one can define a more complex schema to be embedded at that location.

B.2. Using YANG Schema Mount

In this section, we show how a similar result could be attained using YANG Schema Mount. The network-level module is presented in Figure 5.

```
module network-level {
  yang-version 1.1;
  namespace "urn:network-level";
  prefix "net-l";

  import ietf-yang-schema-mount {
    prefix yangmnt;
  }

  container devices {
    list device {
      key device-id;
      leaf device-id {
        type string;
      }
      container device-contents{
        yangmnt:mount-point "device-schema";
      }
    }
  }
}
```

Figure 5: Version of the network-level module using Schema Mount

As explained in Section 3.1, the yang-library corresponding to the modules to embed, as well as the data required by 'ietf-yang-mount' needs to be specified in some other files. Using the 'yanglint' tool from libyang (https://github.com/CESNET/libyang), this module can be compiled to provide a tree representation as shown in Figure 6.

```
module: network-level
  +--rw devices
     +--rw device* [device-id]
        +--rw device-id          string
        +--mp device-contents
           +--rw hostname/    string
           +--ro cpu-usage/?  int8
```

Figure 6: Full tree of both network- and device-level using Schema Mount

The command for obtaining that schema is 'yanglint -f tree -p . -x extension-data.xml -Y network-level-yanglib.xml yang/network-level.yang', assuming all the YANG modules and the two xml files are in the current folder. The file 'network-level-yanglib.xml' contains the YANG Library data for the network-level module. The file 'extension-data.xml' contains the YANG Library data defining the schema to use at the mount point, as well as the data required by YANG Schema Mount. Both are reproduced in Appendix B.3.

B.3. Support Files

The code of the 'device-level' module is given in Figure 7. Then the data files 'network-level-yanglib.xml' and 'extension_data.xml' are provided. These files are needed to compile the Schema Mount version of our example with yanglint.

```
module device-level {
  yang-version 1.1;
  namespace "urn:device-level";
  prefix mnt;

  leaf hostname {
    type string;
    mandatory true;
  }
  leaf cpu-usage {
    type int8;
    config false;
  }
}
```

Figure 7: device-level YANG module

```
<CODE BEGINS> file "network-level-yanglib.xml"
<yang-library xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
     xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
   <module-set>
     <name>main-set</name>
     <module>
       <name>ietf-datastores</name>
       <revision>2018-02-14</revision>
       <namespace>
           urn:ietf:params:xml:ns:yang:ietf-datastores
       </namespace>
     </module>
     <module>
       <name>ietf-yang-library</name>
       <revision>2019-01-04</revision>
```

```
          <namespace>
              urn:ietf:params:xml:ns:yang:ietf-yang-library
          </namespace>
        </module>
        <module>
          <name>ietf-yang-schema-mount</name>
          <revision>2019-01-14</revision>
          <namespace>
              urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount
          </namespace>
        </module>
        <module>
          <name>network-level</name>
          <namespace>urn:network-level</namespace>
        </module>
        <import-only-module>
          <name>ietf-yang-types</name>
          <revision>2013-07-15</revision>
          <namespace>
              urn:ietf:params:xml:ns:yang:ietf-yang-types
          </namespace>
        </import-only-module>
        <import-only-module>
          <name>ietf-inet-types</name>
          <revision>2013-07-15</revision>
          <namespace>
              urn:ietf:params:xml:ns:yang:ietf-inet-types
          </namespace>
        </import-only-module>
      </module-set>
      <schema>
        <name>main-schema</name>
        <module-set>main-set</module-set>
      </schema>
      <datastore>
        <name>ds:running</name>
        <schema>main-schema</schema>
      </datastore>
      <datastore>
        <name>ds:operational</name>
        <schema>main-schema</schema>
      </datastore>
      <content-id>1</content-id>
    </yang-library>
    <modules-state
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
      <module-set-id>2</module-set-id>
    </modules-state>
```

```
   <CODE ENDS>

   <CODE BEGINS> file "extension_data.xml"
   <yang-library xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
                 xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
       <module-set>
           <name>mountee-set</name>
           <module>
               <name>device-level</name>
               <namespace>urn:device-level</namespace>
           </module>
           <module>
               <name>ietf-datastores</name>
               <revision>2018-02-14</revision>
               <namespace>
                   urn:ietf:params:xml:ns:yang:ietf-datastores
               </namespace>
           </module>
           <module>
               <name>ietf-yang-library</name>
               <revision>2019-01-04</revision>
               <namespace>
                   urn:ietf:params:xml:ns:yang:ietf-yang-library
               </namespace>
           </module>
           <import-only-module>
               <name>ietf-yang-types</name>
               <revision>2013-07-15</revision>
               <namespace>
                   urn:ietf:params:xml:ns:yang:ietf-yang-types
               </namespace>
           </import-only-module>
           <import-only-module>
               <name>ietf-inet-types</name>
               <revision>2013-07-15</revision>
               <namespace>
                   urn:ietf:params:xml:ns:yang:ietf-inet-types
               </namespace>
           </import-only-module>
       </module-set>
       <schema>
           <name>test-schema</name>
           <module-set>mountee-set</module-set>
       </schema>
       <datastore>
           <name>ds:running</name>
           <schema>test-schema</schema>
       </datastore>
```

```
        <datastore>
            <name>ds:operational</name>
            <schema>test-schema</schema>
        </datastore>
        <content-id>2</content-id>
    </yang-library>
    <modules-state
        xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
        <module-set-id>2</module-set-id>
    </modules-state>
    <schema-mounts
        xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-schema-mount">
        <mount-point>
            <module>network-level</module>
            <label>device-schema</label>
            <shared-schema/>
        </mount-point>
    </schema-mounts>
    <CODE ENDS>
```

Acknowledgements

Thanks to Ladislav Lhotka and Ignacio Dominguez Martinez-Casanueva for their reviews and comments.

Authors' Addresses

Jean Quilbeuf
Huawei
Email: jean.quilbeuf@huawei.com

Benoit Claise
Huawei
Email: benoit.claise@huawei.com

Thomas Joubert
Huawei
Email: thomas.joubert1@huawei-partners.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 20 April 2024

K. Larsson
Deutsche Telekom

18 October 2023

Mapping YANG Data to Label-Set Time Series
draft-kll-yang-label-tsdb-00

Abstract

This document proposes a standardized approach for representing YANG-modeled configuration and state data, for storage in Time Series Databases (TSDBs) that identify time series using a label-set. It outlines procedures for translating YANG data representations to fit within the label-centric structures of TSDBs and vice versa. This mapping ensures clear and efficient storage and querying of YANG-modeled data in TSDBs.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/plajjan/draft-kll-yang-label-tsdb.

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 20 April 2024.

Table of Contents

1. Introduction 2
2. Specification of the Mapping Procedure 3
2.1. Example: Packet Counters in IETF Interfaces Model 3
2.2. Mapping values 4
2.3. Choice 4
2.4. Host / device name 4
3. Querying YANG modeled time series data 5
3.1. 1. *Basic Queries* 5
3.2. 2. *Filtering by Labels* 5
3.3. 3. *Time-based Queries* 6
3.4. 4. *Aggregations* 6
3.5. 5. *Combining Filters* 6
3.6. 6. *Querying Enumeration Types* 6
4. Requirements on time series databases 7
4.1. Support for String Values 7
4.2. Sufficient Path Length 7
4.3. High Cardinality 8
5. Normative References 8
Author's Address 8

1. Introduction

The aim of this document is to define rules for representing configuration and state data defined using the YANG data modeling language [RFC7950] as time series using a label-centric model.

The majority of modern Time Series Databases (TSDBs) employ a label-centric model. In this structure, time series are identified by a set of labels, each consisting of a key-value pair. These labels facilitate efficient querying, aggregation, and filtering of data over time intervals. Such a model contrasts with the hierarchical nature of YANG-modeled data. The challenge, therefore, lies in ensuring that YANG-defined data, with its inherent structure and depth, can be seamlessly integrated into the flat, label-based structure of most contemporary TSDBs.

This document seeks to bridge this structural gap, laying out rules and guidelines to ensure that YANG-modeled configuration and state data can be effectively stored, queried, and analyzed within label-centric TSDBs.

2. Specification of the Mapping Procedure

Instances of YANG data nodes are mapped to metrics. Only nodes that carry a value are mapped. This includes leafs and presence containers. The hierarchical path to a value, including non-presence containers and lists, form the path that is used as the name of the metric. The path is formed by joining YANG data nodes using _. Special symbols, e.g. -, in node names are replaced with _.

List keys are mapped into labels. The path to the list key is transformed in the same way as the primary name of the metric. Compound keys have each key part as a separate label.

2.1. Example: Packet Counters in IETF Interfaces Model

Consider the in-unicast-pkts leaf from the IETF interfaces model that captures the number of incoming unicast packets on an interface:

Original YANG Instance-Identifier: yang /interfaces/interface[name='eth0']/statistics/in-unicast-pkts

Following the mapping rules defined:

1. The path components, including containers and list names, are transformed into the metric name by joining the node names with _. Special symbols, e.g. - are replaced with _.

Resulting Metric Name: interfaces_interface_statistics_in_unicast_pkts

1. The list key "predicate", which in this case is the interface name (eth0), is extracted and stored as a separate label. The label key represents the complete path to the key.

Resulting Label: interfaces_interface_name = eth0

1. The leaf value, which represents the actual packet counter, remains unchanged and is directly mapped to the value in the time series database.

For instance, if the packet counter reads 5,432,100 packets:

Value: 5432100

1. As part of the standard labels, a server identification string is also included. A typical choice of identifier might be the hostname. For this example, let's assume the device name is router-01:

Label: host = router-01

Final Mapping in the TSDB:

* Metric: interfaces_interface_statistics_in_unicast_pkts
* Value: 5432100
* Labels:
  - host = router-01
  - interfaces_interface_name = eth0

2.2. Mapping values

Leaf values are mapped based on their intrinsic type:

* All integer types are mapped to integers and retain their native representation
  - some implementations only support floats for numeric values
* decimal64 values are mapped to floats and the value should be rounded and truncated as to minimize the loss of information
* Enumeration types are mapped using their string representation.
* String types remain unchanged.

2.3. Choice

Choice constructs from YANG are disregarded and not enforced during the mapping process. Given the temporal nature of TSDBs, where data spans across time, different choice branches could be active in a single data set, rendering validation and storage restrictions impractical.

2.4. Host / device name

There is an implicit host label identifying the server, typically set to the name of the host originating the time series data.

Instance data retrieved from YANG-based servers do not generally identify the server it originates from. As a time series database is likely going to contain data from multiple servers, the host label is used to identify the source of the data.

3. Querying YANG modeled time series data

The process of storing YANG-modeled data in label-centric TSDBs, as defined in the previous sections, inherently structures the data in a way that leverages the querying capabilities of modern TSDBs. This chapter provides guidelines on how to construct queries to retrieve this data effectively.

3.1. 1. *Basic Queries*

To retrieve all data points related to incoming unicast packets from the IETF interfaces model:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts

3.2. 2. *Filtering by Labels*

To retrieve incoming unicast packets specifically for the interface eth0:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE interfaces_interface_name = 'eth0'

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts{interfaces_interface_name="eth0"}

Similarly, to filter by device / host name:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE host = 'router-01'

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts{host="router-01"}

3.3. 3. *Time-based Queries*

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE time > now() - 24h

Prometheus fetches data based on the configured scrape interval and retention policies, so time-based filters in PromQL often center around the range vectors. For data over the last 24 hours:

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts[24h]

3.4. 4. *Aggregations*

To get the average number of incoming unicast packets over the last hour:

* *InfluxQL*: sql SELECT MEAN(value) FROM interfaces_interface_statistics_in_unicast_pkts WHERE time > now() - 1h GROUP BY time(10m)

* *PromQL*: promql avg_over_time(interfaces_interface_statistics_in_unicast_pkts[1h])

3.5. 5. *Combining Filters*

To retrieve the sum of incoming unicast packets for eth0 on router-01 over the last day:

* *InfluxQL*: sql SELECT SUM(value) FROM interfaces_interface_statistics_in_unicast_pkts WHERE interfaces_interface_name = 'eth0' AND host = 'router-01' AND time > now() - 24h

* *PromQL*: promql sum(interfaces_interface_statistics_in_unicast_pkts{interfaces_interface_name="eth0", host="router-01"})[24h]

3.6. 6. *Querying Enumeration Types*

In YANG models, enumerations are defined types with a set of named values. The oper-status leaf in the IETF interfaces model is an example of such an enumeration, representing the operational status of an interface.

For instance, the oper-status might have values such as up, down, or testing.

To query interfaces that have an oper-status of up:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_oper_status WHERE value = 'up'

* *PromQL*: promql interfaces_interface_oper_status{value="up"}

Similarly, to filter interfaces with oper-status of down:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_oper_status WHERE value = 'down'

* *PromQL*: promql interfaces_interface_oper_status{value="down"}

This approach allows us to effectively query interfaces based on their operational status, leveraging the enumeration mapping within the TSDB.

4. Requirements on time series databases

This document specifies a mapping to a conceptual representation, not a particular concrete interface. To effectively support the mapping of YANG-modeled data into a label-centric model, certain requirements must be met by the Time Series Databases (TSDBs). These requirements ensure that the data is stored and retrieved in a consistent and efficient manner.

4.1. Support for String Values

Several YANG leaf types carry string values, including the string type itself and all its descendants as well as enumerations which are saved using their string representation.

The chosen TSDB must support the storage and querying of string values. Not all TSDBs inherently offer this capability, and thus, it's imperative to ensure compatibility.

4.2. Sufficient Path Length

YANG data nodes, especially when representing deep hierarchical structures, can result in long paths. When transformed into metric names or labels within the TSDB, these paths might exceed typical character limits imposed by some databases. It's essential for the TSDB to accommodate these potentially long names to ensure data fidelity and avoid truncation or loss of information.

4.3. High Cardinality

Given the possibility of numerous unique label combinations (especially with dynamic values like interface names, device names, etc.), the chosen TSDB should handle high cardinality efficiently. High cardinality can impact database performance and query times, so it's essential for the TSDB to have mechanisms to manage this efficiently.

5. Normative References

[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>.

Author's Address

Kristian Larsson
Deutsche Telekom
Email: kristian@spritelink.net

IVY Working Group
Internet-Draft
Intended status: Standards Track
Expires: 18 April 2024

T. Li
R. Bonica
Juniper Networks

16 October 2023

A YANG model for Power Management
draft-li-ivy-power-01

Abstract

Network sustainability is a key issue facing the industry. Networks consume significant amounts of power at a time when the cost of power is rising and sensitivity about sustainability is very high. As an industry, we need to find ways to optimize the power efficiency of our networks both at a micro and macro level. We have observed that traffic levels fluctuate and when traffic ebbs there is much more capacity than is needed. Powering off portions of network elements could save a significant amount of power, but to scale and be practical, this must be automated.

The natural mechanism for enabling automation would be a Yet Another Next Generation (YANG) interface, so this document proposes a YANG model for power management.

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 18 April 2024.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Requirement Language 3
2. Power Management Elements 3
2.1. Power consumption 3
2.2. Power control capability 4
2.3. Power control 4
2.4. Automatic Power Management 4
3. Functional Dependencies 4
3.1. Required Components 5
3.2. Dependent components 5
4. Tree Representation 5
5. Traffic Planning 5
5.1. Tree Representation 6
6. Security Considerations 6
7. IANA Considerations 6
8. Normative References 6
Authors' Addresses 7

1. Introduction

Network sustainability is a key issue facing the industry. Networks consume significant amounts of power at a time when the cost of power is rising and sensitivity about sustainability is very high. As an industry, we need to find ways to optimize the power efficiency of our networks both at a micro and macro level. We have observed that traffic levels fluctuate and when traffic ebbs there is much more capacity than is needed. Powering off portions of network elements could save a significant amount of power, but to scale and be practical, this must be automated.

The natural mechanism for enabling automation would be a Yet Another Next Generation (YANG) interface, so this document proposes a YANG model for power management.

[RFC8348] already provides a model for server hardware management, but does not naturally extend to routers and other network elements. That gap is currently being addressed by

[I-D.wzwb-opsawg-network-inventory-management] and [I-D.ietf-ccamp-network-inventory-yang]. This document extends the work presented there to include power management. Specifically, this document augments the 'component' object found at /ietf-network-hardware-inventory/network-hardware-inventory/network-elements/network-element/components/component in [I-D.ietf-ccamp-network-inventory-yang].

This initial draft only provides a tree representation. When there is rough consensus on the tree represetnation, the details of the model will be fleshed out.

1.1. Requirement Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

2. Power Management Elements

The models mentioned above already model a router or network element as a set of components. The details of those components are left to the specific implementation and can be at any level of specificity. Thanks to this flexibility, it is necessary and sufficient that we characterize power management relative to components.

The elements defined below allow management entities to understand how much power each component is using and whether the component can be placed into a 'power-save' mode where it would consume less power. Another element allows the management plane to put the component into power-save mode.

2.1. Power consumption

```
Name: used-power
Node Type: leaf
Data Type: uint32
Description: Power drawn by the component, in watts.
```

This node is applied to components in the model. If an accurate dynamic power measurement is not available, then static power estimates are acceptable.

2.2. Power control capability

Name: power-save-capable
Node Type: leaf
Data Type: boolean
Description: True if the component can be put into power-save mode.

2.3. Power control

Name: power-save
Node Type: leaf
Data Type: Boolean
Access: Read/write
Description: True if the component is in power-save mode.

2.4. Automatic Power Management

Some systems (e.g., fan trays) have the capability to self-manage their power consumption. However there are cases where this capability should be disabled.

Name: automatic-power-management
Node Type: leaf
Data Type: Boolean
Access: Read/write
Description: True if the component is regulating its own power.

3. Functional Dependencies

Most inventory models have a hierarchy of components. This hierarchy reflects the physical structure of the system (e.g., a line card can physically contain a port).

With regard to physical containment, components maintain a one-to-many relationship. That is, Component A can contain many other components, including Component B. However, component B can be contain by only one component (i.e., Component A.)

However, legacy inventory models do not reflect functional dependencies. Specifically, they do not indicate which components obtain services from, and therefore depend, components other than their container. Because funtional dependencies are relavant to power management, they are included in the proposed model.

With regard to functional dependencies, components maintain a many-to-many relationship. That is, a component can reuire on many components and be required by many other components.

Functional dependencies may be updated dynamically.

3.1. Required Components

This container holds a list of components that the component uses. For example, a linecard uses a set of switch cards, so the switch cards would be required components. If the bandwidth used by the linecard changes, then the set of switch cards that are required may change dynamically.

Name: required-components
Node Type: list
Description: A list of other components that are required for this component to operate.

3.2. Dependent components

This container holds a list of components that are used by this component. For example, a switch card is used by a set of line cards, so the line cards would be dependent components. This list can also change dynamically.

Name: dependent-components
Node Type: list
Description: A list of other components that are used by this component.

4. Tree Representation

```
+--ro component* [uuid]
   +--ro uuid                          yang:uuid
   +--ro used-power?                   uint32
   +--ro power-save-capable?           boolean
   +--rw power-save?                   boolean
   +--ro required-components*          -> ../uuid
   +--ro dependent-components*         -> ../uuid
```

5. Traffic Planning

Some systems have the capability of automatically managing their power consumption if they have an understanding of the expected traffic loads. To provide this, we provide the expected input and output bandwidth for each interface and augment [RFC7223] with the following:

```
          Name: expected-input-bandwidth
          Node Type: leaf
          Data Type: uint32
          Default: 0
          Access: Read/write
          Description: The expected input bandwidth of the interface,
            in Mbps. A value of zero (0) indicates full bandwidth.

          Name: expected-output-bandwidth
          Node Type: leaf
          Data Type: uint32
          Default: 0
          Access: Read/write
          Description: The expected output bandwidth of the interface,
            in Mbps. A value of zero (0) indicates full bandwidth.
```

5.1. Tree Representation

```
   +--rw interface* [name]
      +--rw expected-input-bandwidth?     uint32
      +--rw expected-output-bandwidth?    uint32
```

6. Security Considerations

YANG provides information about and configuration capabilities to the network management plane. Other mechanisms already exist that help secure these interactions. This document extends the scope of what can be controlled by the management plane, but creates no new access paths.

7. IANA Considerations

This document makes no requests for IANA.

8. Normative References

[RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <https://www.rfc-editor.org/info/rfc8348>.

[I-D.wzwb-opsawg-network-inventory-management] Wu, B., Zhou, C., Wu, Q., and M. Boucadair, "A Network Inventory Management Model", Work in Progress, Internet-Draft, draft-wzwb-opsawg-network-inventory-management-03, 24 July 2023, <https://datatracker.ietf.org/doc/html/draft-wzwb-opsawg-network-inventory-management-03>.

[I-D.ietf-ccamp-network-inventory-yang]
Yu, C., Belotti, S., Bouquier, J., Peruzzini, F., and P. Bedard, "A YANG Data Model for Network Hardware Inventory", Work in Progress, Internet-Draft, draft-ietf-ccamp-network-inventory-yang-02, 9 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ccamp-network-inventory-yang-02>.

[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>.

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

Authors' Addresses

Tony Li
Juniper Networks
Email: tony.li@tony.li

Ron Bonica
Juniper Networks
Email: rbonica@juniper.net

NETMOD
Internet-Draft
Intended status: Standards Track
Expires: 22 April 2024

J. Lindblad
Cisco

20 October 2023

Philatelist, YANG-based collection and aggregation framework integrating Telemetry data and Time Series Databases
draft-lindblad-tlm-philatelist-00

Abstract

Timestamped telemetry data is collected en masse today. Mature tools are typically used, but the data is often collected in an ad hoc manner. While the dashboard graphs look great, the resulting data is often of questionable quality, not well defined, and hard to compare with seemingly similar data from other organizations.

This document proposes a standard, extensible, cross domain framework for collecting and aggregating timestamped telemetry data in a way that combines YANG, metadata and Time Series Databases to produce more dependable and comparable results.

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://janlindblad.github.io/netmod-tlm-philatelist/draft-lindblad-tlm-philatelist.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-lindblad-tlm-philatelist/.

Source for this draft and an issue tracker can be found at https://github.com/janlindblad/netmod-tlm-philatelist.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
  1.1. The Problem 3
  1.2. The Solution 3
  1.3. The Philatelist Name 4
2. Conventions and Definitions 4
3. Architecure Overview 5
  3.1. The Provider Component 7
  3.2. The Collector Component 8
  3.3. The Processor and Aggregator Components 10
4. YANG-based Telemetry Outlook 13
5. YANG Modules 13
  5.1. Base types module for Philatelist 13
  5.2. Provider interface module for Philatelist 21
  5.3. Collector interface module for Philatelist 23
  5.4. Aggregator interface module for Philatelist 27
6. Security Considerations 30
7. IANA Considerations 30
8. References 30
  8.1. Normative References 30
  8.2. Informative References 31
Acknowledgments 31
Author's Address 31

1. Introduction

1.1. The Problem

Many organizations today are collecting large amounts of telemetry data from their networks and data centers for a variety of purposes. Much (most?) of this data is funneled into a Time Series Database (TSDB) for display in a dashboard or further (AI-backed) processing and decision making.

While this data collection is often handled using standard tools, there generally seems to be little commonality when it comes to what is meaured, how the data is aggregated, or definitions of the measured quantities (if any).

Data science issues like adding overlapping quantities, adding quantities of different units of measurement, or quantities with different scopes, are likely common. Such errors are hard to detect given the ad hoc nature of the collection. This often leads to uncertainty regarding the quality of the conclusions drawn from the collected data.

1.2. The Solution

The Philatelist framework proposes to standardize the collection, definitions of the quantities measured and meta data handling to provide a robust ground layer for telemetry collection. The architecture defines a few interfaces, but allows great freedom in the implementations with its plug-in architecture. This allows flexibility enough that any kind of quantitiy can be measured, any kind of collection protocol and mechanism employed, and the data flows aggregated using any kind of operation.

To do this, YANG is used both to describe the quantities being measured, as well as act as the framework for the metadata management. Note that the usa of YANG here does not limit the architecture to traditional YANG-based transport protocols. YANG is used to describe the data, regardless of which format it arrives in.

Initially developed in context of the Power and Energy Efficiency work (POWEFF), we realized both the potential and the need for this collection and aggregation architecture to become a general framework for collection of a variety of metrics.

There is not much point in knowing the "cost side" of a running system (as in energy consumption or CO2-emissions) if that cannot be weighed against the "value side" delivered by the system (as in transported bytes, VPN connections, music streaming hours, or number of cat videos, etc.), which means traditional performance metrics will play an equally important role in the collection.

In this initial version, we have done nothing to pull the proposed YANG modules out of its POWEFF roots and generalize it for general telemetry. We believe the ideas and merits of this framework architecture will be apparent nonetheless in this first version. For the next version, we certainly need to generalize the quantities measured and rename the YANG modules and node names.

1.3. The Philatelist Name

This specification is about a framework for collection, aggregation and interpretation of timestamped telemetry data. The definition of "philatelist" seems close enough.

1. philatelist

noun. [’flætlst’] a collector and student of postage stamps.

Synonyms
- collector
- aggregator

Figure 1: Source: https://www.synonym.com/synonyms/philatelist

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.

This document uses the terminology defined in [RFC7950].

In addition, this document defines the following terms:

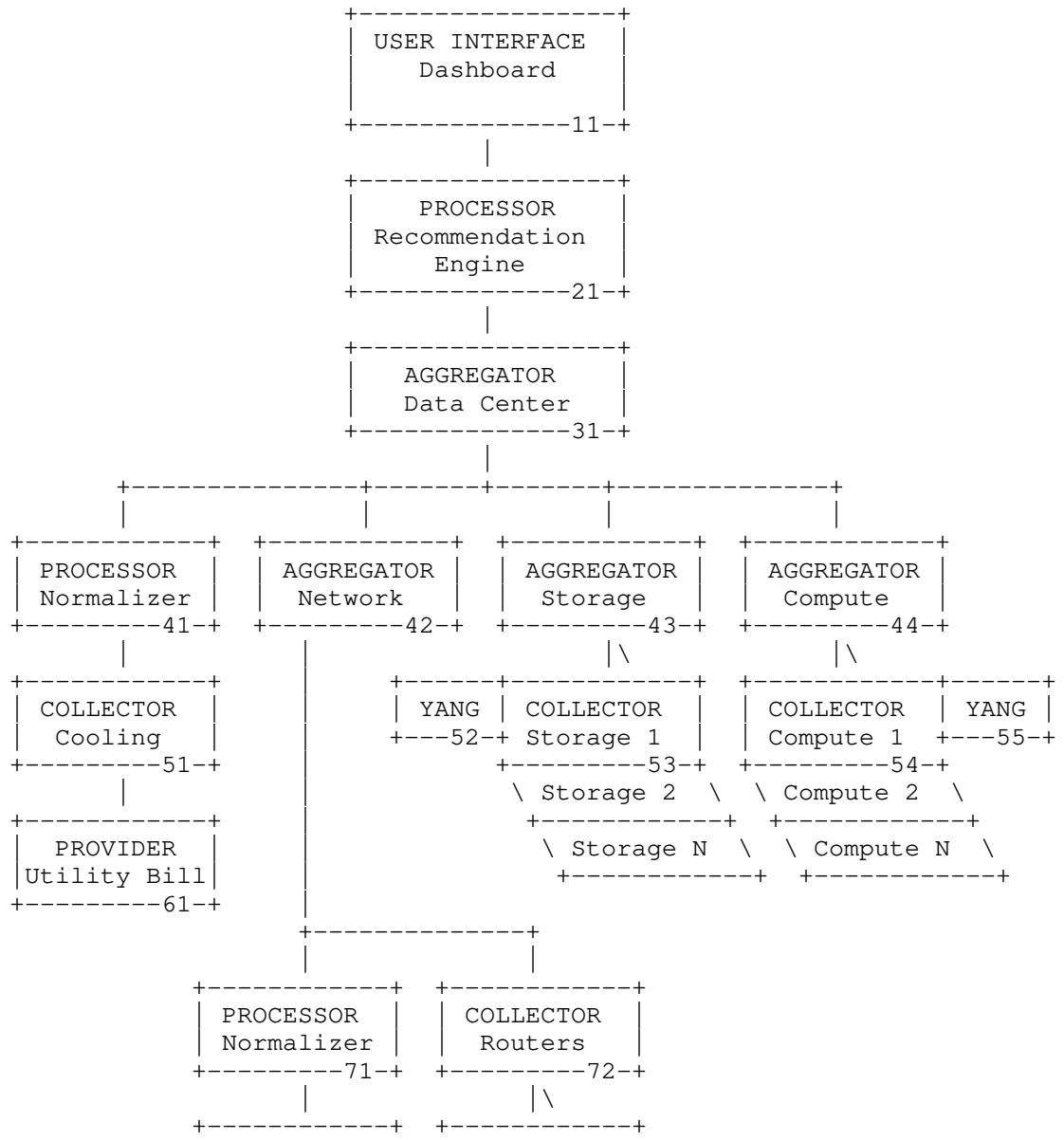
TSDB Time Series Database.

Sensor An entity in a system that delivers a snapshot value of some quantity pertaining to the system. Sensors are identified by their Sensor Path.

Sensor Path A textual representation of the sensor’s address within the system.

3. Architecure Overview

The deployment of a Philatelist framework consists of a collection of plug-in compomnents with well defined interfaces. Here is an example of a deployment. Each box is numbered in the lower right for easy reference.



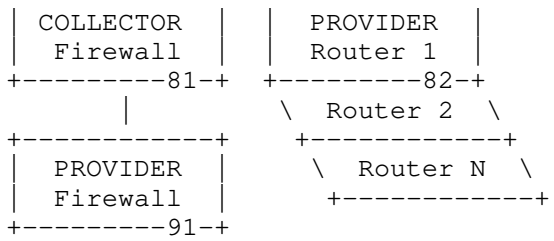


Figure 2: Example Philatelist component deployment.

Each component in the above diagram, represents a logical function. Many boxes could be running within a single server, or they could be fully distrubuted, or anything in between.

Provider components (61, 82, 91) are running on a telemetry source system that supports a YANG-based telemetry data server. The telemetry data flows from the telemetry source system to a Time Series Database (TSDB).

Collector components (51, 72, 81) ensure the Providers are programmed properly to deliver the telemetry data to the TSDB designated by the collector. In some cases this flow may be direct from the source to the TSDB, in other cases, it may be going through the collector. In some cases the collector may be polling the source, in others it may have set up an automatic, periodic subscription.

Many telemetry source systems will not have any on-board YANG-based telemetry server. Such servers will instead be managed by a collector specialized to handle a particular kind of source server (53, 54). These specialized collectors are still responsible to set up a telemetry data stream from them to the collector's TSDB. In this case, the collector will also supply a YANG description (52, 55) of the incoming data stream.

Processor components (21, 41, 71) are transforming the data stream in some way, e.g. converting from one unit of measurement to another, or adjusting the data values recorded to also include some aspect that this particular sensor is not taking into account.

Aggregator components (31, 42, 43, 44) combine the time series telemetry data flows using some operation, e.g. summing, averaging or computing the max or min over them. In this example there are aggregators for Network, Storage, Compute and the entire Data Center

On top of the stack, we may often find a (graphical) user interface (11), for human consumption of the intelligence acquired by the system. Equally relevant is of course an (AI) application making decisions based on findings in the aggregated telemetry flow.

3.1. The Provider Component

A Provider is a source of telemetry data that also offers a YANG-based management interface. Each provider typically has a large number of "sensors" that can be polled or in some cases subscribed to.

One problem with the sensors is that they are spread around inside the source system, and may not be trivial to locate. Also, the metadata asscociated with the sensor is often only missing or only available in human readable form (free form strings), rather than in a strict machine parsable format.

```
/hardware/component[name="psu3"]/.../sensor-data/value
...
/interfaces/interface[name="eth0/0"]/.../out-broadcast-packets
...
/routing/mpls/mpls-label-blocks/.../inuse-labels-count
...
```

Figure 3: Example of scattered potential sensors in a device.

To solve these problems, the Provider YANG module contains a sensor-catalog list. Essentially a list of all interesting sensors available on the system, with their sensor paths and machine parsable units, definition and any other metadata.

An admin user or application can then copy the sensor definition from the sensor catalog and insert into the configuration in the colletor.

```
  +--ro sensor-catalog
      +--ro sensors
        +--ro sensor* [path]
            +--ro path?                    xpath
            +--ro sensor-type?             identityref
            +--ro sensor-location?         something
            +--ro sensor-state?            something
            +--ro sensor-current-reading?  something
            +--ro sensor-precision?        string
```

Figure 4: YANG tree diagram of the Provider sensor-catalog.

Note: The "something" YANG-type is used in many places in this document. That is just a temporarty placeholder we use until we have figured out what the appropriate type should be.

The sensor types are defined as YANG identities, making them maximally extensible. Examples of sensor types might be energy measured in kWh, or energy measured in J, or temperature measured in F, or in C, or in K.

3.2. The Collector Component

Collector components collect data points from sources, typically by periodic polling or subscriptions, and ensure the collected data is stored in a Time Series Database (TSDB). The actual data stream may or may not be passing through the collector component; the collector is responsible for ensuring data flows from the source to the destination TSDB and that the data has a YANG description and is tagged with necessary metadata. How the collector agrees with a source to deliver data in a timely manner is beyond the scope of this document.

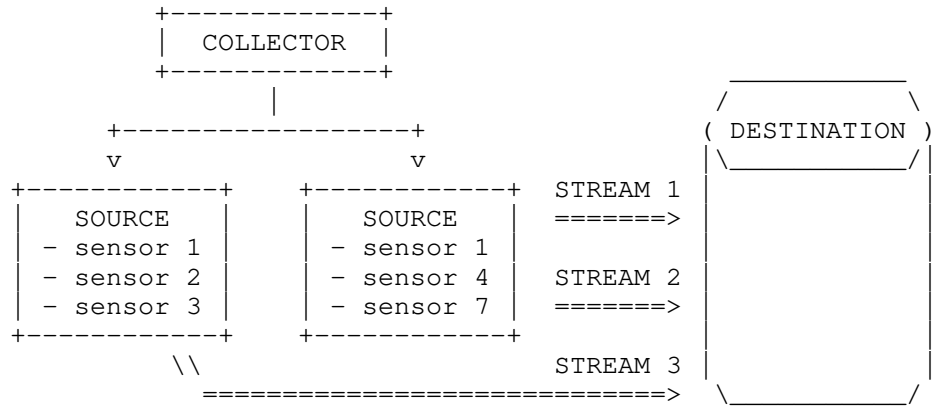


Figure 5: Example of Collector setting up three streams of telemetry data from two sources to one desitination.

Each source holds a number of sensors that may be queried or subscribed to. The collector arranges the sensors into sensour groups that presumably are logically related, and that are collected using the same method. A number of collection methods (some YANG-based, some not) are modeled directly in the ietf-poweff-collector.yang module, but the set is designed to be easily extensible.

```
+-- sensor-groups
|  +-- sensor-group* [id]
|     +-- id?                                  something
|     +-- method?                              identityref
|     +-- get-static-url-once
|     |  +-- url?                              something
|     |  +-- format?                           something
|     +-- gnmi-polling
|     |  +-- encoding?                         something
|     |  +-- protocol?                         something
|     +-- restconf-get-polling
|     |  +-- xxx?                              something
|     +-- netconf-get-polling
|     |  +-- xxx?                              something
|     +-- restconf-yang-push-subscription
|     |  +-- xxx?                              something
|     +-- netconf-yang-push-subscription
|     |  +-- xxx?                              something
|     +-- redfish-polling
|     |  +-- xxx?                              something
|     +-- frequency?                           sample-frequency
|     +-- path* [path]
|        +-- path?                             xpath
|        +-- sensor-type?                      identityref
+-- streams
  +-- stream* [id]
      +-- id?                                  something
      +-- source*                              string
      +-- sensor-group* [name]
      |  +-- name?   -> ../../../sensor-groups/sensor-group/id
      +-- destination?    -> ../../../destinations/destination/id
```

Figure 6: YANG tree diagram of the Collector sensor-groups and streams.

The sensor groups are then arranged into streams from a collection of sources (that support the same set of sensor groups) to a destination. This structure has been chosen with the assumption that there will be many source devices with the same set of sensor groups, and we want to minimize repetition.

3.3. The Processor and Aggregator Components

Processor components take an incoming data flow and transforms it somehow, and possibly augments it with a flow of derived information. The purpose of the transformation could be to convert between different units of measurement, correct for known errors in in the input data, or fill in approximate values where there are holes in the input data.

Aggregator components take multiple incoming data flows and combine them, typically by adding them together, taking possible differences in cadence in the input data flows into account.

Processor and Aggregator components provide a YANG model of the output data, just like the Collector components, so that all data flowing in the system has a YANG description and is associated with metadata.

Note: In the current version of the YANG modules, a Processor is simply an Aggregator with a single input and output. Unless we see a need to keep these two component types separate, we might remove the Processor component and keep it baked in with the Aggregator.

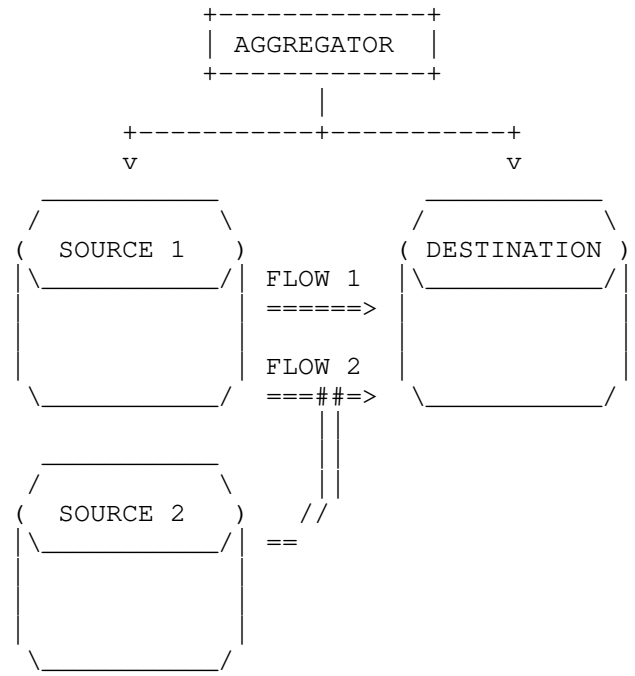


Figure 7: Example of an Aggregator setting up two flows of telemetry data from two sources to one desitination.

In this diagram, the sources and destination look like separate TSDBs, which they might be. They may also be different buckets within the same TSDB.

Each flow is associated with one or more inputs, one output and a series of processing operations. Each input flow and output flow may have an pre-processing or post-processing operation applied to it separately. Then all the input flows are combined using one or more aggregation operations. Some basic operations have been defined in the Aggregator YANG module, but the set of operations has been designed to be maximally extensible.

```
+-- flows
|  +-- flow* [id]
|     +-- id?                                  string
|     +-- (chain-position)?
|        +--:(input)
|        |  +-- input
|        |     +-- source?
|        |           -> ../../../../../sources/source/id
|        +--:(output)
|        |  +-- output
|        |     +-- destination?
|        |           -> ../../../../../destinations/destination/id
|        +--:(middle)
|           +-- middle
|              +-- inputs*
|              |     -> ../../../../flows/flow/id
|              +-- pre-process-inputs?
|              |     -> ../../../../operations/operation/id
|              +-- aggregate?
|              |     -> ../../../../operations/operation/id
|              +-- post-process-output?
|                    -> ../../../../operations/operation/id
+-- operations
  +-- operation* [id]
     +-- id?                                   something
     +-- (op-type)?
       +--:(linear-sum)
       |  +-- linear-sum
       +--:(linear-average)
       |  +-- linear-average
       +--:(linear-max)
       |  +-- linear-max
       +--:(linear-min)
       |  +-- linear-min
       +--:(rolling-average)
       |  +-- rolling-average
       |     +-- timespan?                   something
       +--:(filter-age)
       |  +-- filter-age
       |     +-- min-age?                    something
       |     +-- max-age?                    something
       +--:(function)
           +-- function
             +-- name?                       something
```

Figure 8: YANG tree diagram of the Aggregator flows and operations.

The operations listed above are basic aggregation operations. Linear-sum is just adding all the input sources together, with linear interpolation when their data points don't align perfectly in time. Rolling average is averaging the input flows over a given length of time. The filter-age drops all data points that are outside the min to max age. The function allows plugging in any other function the Aggregator may have defined, but more importantly, the operations choice is easily extended using YANG augment to include any other IETF or vendor specified extensions.

4. YANG-based Telemetry Outlook

Much work has already gone into the area of telemetry, YANG, and even their intersection. E.g. [I-D.draft-ietf-opsawg-collected-data-manifest-01] and [I-D.draft-claise-netconf-metadata-for-collection-03] come to mind.

Even though this work has a solid foundation and shares many or most of the goals with this work, we (the POWEFF team) have not found it easy to apply the above work directly in the practical work we do. So what we have tried to do is a very pragmatic approach to telemetry data collection the way we see it happening on the ground combined with the benefits of Model Driven Telemetry (MDT), in practice meaning YANG-based with additional YANG-modeled metadata.

Many essential data sources in real world deployments do not support any YANG-based interfaces, and that situation is expected to remain for the forseable future, which is why we find it important to be able to ingest data from free form (often REST-based) interfaces, and then add the necessary rigor on the Collector level. Then output the datastreams in formats that existing, mature tools can consume directly.

In particular, this draft depends on the mapping of YANG-based structures to the typical TSDB tag-based formats described in [I-D.draft-kll-yang-label-tsdb-00].

For the evolution of the YANG-based telemetry area, we believe this approach, combining pragmatism in the data flow interfaces with rigor regarding the data content, is key. We would like to make this work fit in with the works of others in the field.

5. YANG Modules

5.1. Base types module for Philatelist

```
<CODE BEGINS>
module ietf-poweff-types {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-types";
  prefix ietf-poweff-types;

  organization
    "IETF OPSA (Operations and Management Area) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>
     Editor:  Jan Lindblad
              <mailto:jlindbla@cisco.com>
     Editor:  Snezana Mitrovic
              <mailto:snmitrov@cisco.com>
     Editor:  Marisol Palmero
              <mailto:mpalmero@cisco.com>";
  description
    "This YANG module defines basic quantities, measurement units
    and sensor types for the POWEFF framework.

     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
     (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 2023-10-12 {
    description
      "Initial revision of POWEFF types";
    reference
      "RFC XXXX: ...";
  }

  typedef something { // FIXME: Used when we haven't decided the type yet
    type string;
  }
  typedef xpath {
    type string; // FIXME: Proper type needed
```

```
  }
  typedef sample-frequency {
    type string; // FIXME: Proper type needed
  }

  // ========== SENSOR-CLASS ==============================
  identity sensor-class {
    description "Sensor's relation to the asset it sits on.";
  }
  identity sc-input {
    base sensor-class;
    description "Sensor reports input quantity of the asset it sits
      on.";
  }
  identity sc-output {
    base sensor-class;
    description "Sensor reports output quantity of the asset it sits
      on.";
  }
  identity sc-allocated {
    base sensor-class;
    description "Sensor reports (maximum) allocated quantity of the
      asset it sits on.";
  }

  // ========== SENSOR-QUANTITY ===========================
  identity sensor-quantity {
    description "Sensor's quantity being measured.";
  }
  identity sq-voltage {
    base sensor-quantity;
    description "Sensor reports electric tension, voltage.";
  }
  identity sq-current {
    base sensor-quantity;
    description "Sensor reports electric current.";
  }
  identity sq-power {
    base sensor-quantity;
    description "Sensor reports power draw (energy per unit of time).";
  }
  identity sq-power-apparent {
    base sq-power;
    description "Sensor reports apparent power, i.e. average electrical
      current times voltage (in VA).";
  }
  identity sq-power-true {
    base sq-power;
```

```
      description "Sensor reports true power, i.e. integral over current
        and voltage at each instant in time.";
    }
    identity sq-energy {
      base sensor-quantity;
      description "Sensor reports actual energy drawn by asset.";
    }
    identity sq-co2-emission {
      base sensor-quantity;
      description "Sensor reports CO2 (carbon dioxide) emission by
        asset.";
    }
    identity sq-co2eq-emission {
      base sensor-quantity;
      description "Sensor reports CO2 (carbon dioxide) equivalent
        emission by asset.";
    }
    identity sq-temperature {
      base sensor-quantity;
      description "Sensor reports temperature of asset.";
    }

    // ========== SENSOR-UNIT ==============================
    identity sensor-unit {
      description "Sensor's unit of reporting.";
    }
    identity su-volt {
      base sensor-unit;
      base sq-voltage;
      description "Sensor unit volt, V.";
    }
    identity su-ampere {
      base sensor-unit;
      base sq-current;
      description "Sensor unit ampere, A.";
    }
    identity su-watt {
      base sensor-unit;
      base sq-power;
      description "Sensor unit watt, W.";
    }
    identity su-voltampere {
      base sensor-unit;
      base sq-power;
      description "Sensor unit Volt*Ampere, VA.";
    }
    identity su-kw {
      base sensor-unit;
```

```
      base sq-power;
      description "Sensor unit kilowatt, kW.";
    }
    identity su-joule {
      base sensor-unit;
      base sq-energy;
      description "Sensor unit joule, J.";
    }
    identity su-wh {
      base sensor-unit;
      base sq-energy;
      description "Sensor unit watthour, Wh.";
    }
    identity su-kwh {
      base sensor-unit;
      base sq-energy;
      description "Sensor unit kliowatthour, kWh.";
    }
    identity su-kelvin {
      base sensor-unit;
      base sq-temperature;
      description "Sensor unit kelvin, K.";
    }
    identity su-celsius {
      base sensor-unit;
      base sq-temperature;
      description "Sensor unit celsius, C.";
    }
    identity su-farenheit {
      base sensor-unit;
      base sq-temperature;
      description "Sensor unit farenheit, F.";
    }
    identity su-gram {
      base sensor-unit;
      base sq-co2-emission;
      description "Sensor unit gram, g.";
    }
    identity su-kg {
      base sensor-unit;
      base sq-co2-emission;
      description "Sensor unit kliogram, kg.";
    }
    identity su-ton {
      base sensor-unit;
      base sq-co2-emission;
      description "Sensor unit ton, t.";
    }
```

```
// ========== SENSOR-TYPE ==============================
identity sensor-type {
  description "Sensor's type, i.e. combination of class, quantity and
    unit.";
}
identity st-v-in {
  base sensor-type;
  base sc-input;
  base sq-voltage;
  base su-volt;
  description "Sensor reporting Voltage In to asset.";
}
identity st-v-out {
  base sensor-type;
  base sc-output;
  base sq-voltage;
  base su-volt;
  description "Sensor reporting Voltage Out of asset.";
}
identity st-i-in {
  base sensor-type;
  base sc-input;
  base sq-current;
  base su-ampere;
  description "Sensor reporting Current In to asset.";
}
identity st-i-out {
  base sensor-type;
  base sc-output;
  base sq-current;
  base su-ampere;
  description "Sensor reporting Current Out of asset.";
}
identity st-p-in-apparent-watt {
  base sensor-type;
  base sc-input;
  base sq-power-apparent;
  base su-voltampere;
  description "Sensor reporting Power In to asset as apparent (I*U)
    power.";
}
identity st-p-out-apparent-watt {
  base sensor-type;
  base sc-output;
  base sq-power-apparent;
  base su-voltampere;
  description "Sensor reporting Power Out of asset as apparent (I*U)
    power.";
```

```
    }
    identity st-p-in-true-watt {
      base sensor-type;
      base sc-input;
      base sq-power-true;
      base su-watt;
      description "Sensor reporting Power In to asset as true power.";
    }
    identity st-p-out-true-watt {
      base sensor-type;
      base sc-output;
      base sq-power-true;
      base su-watt;
      description "Sensor reporting Power Out of asset as true power.";
    }
    identity st-p-allocated-watt {
      base sensor-type;
      base sc-allocated;
      base sq-power;
      base su-watt;
      description "Sensor reporting Allocated Power for asset.";
    }
    identity st-w-j {
      base sensor-type;
      base sq-energy;
      base su-joule;
      description "Sensor reporting energy draw of asset in J.";
    }
    identity st-w-wh {
      base sensor-type;
      base sq-energy;
      base su-wh;
      description "Sensor reporting energy draw of asset in Wh.";
    }
    identity st-w-kwh {
      base sensor-type;
      base sq-energy;
      base su-kwh;
      description "Sensor reporting energy draw of asset in kWh.";
    }
    identity st-t-k {
      base sensor-type;
      base sq-temperature;
      base su-kelvin;
      description "Sensor reporting Temperature of asset in K.";
    }
    identity st-t-c {
      base sensor-type;
```

```
      base sq-temperature;
      base su-celsius;
      description "Sensor reporting Temperature of asset in °C.";
    }
    identity st-t-f {
      base sensor-type;
      base sq-temperature;
      base su-farenheit;
      description "Sensor reporting Temperature of asset in °F.";
    }

    // ========== COLLECTION-METHOD ==============================

    identity collection-method;
    identity cm-polled {
      base collection-method;
    }
    identity cm-gnmi {
      base collection-method;
    }
    identity cm-restconf {
      base collection-method;
    }
    identity cm-netconf {
      base collection-method;
    }
    identity cm-redfish {
      base collection-method;
    }
    identity get-static-url-once {
      base collection-method;
    }
    identity gnmi-polling {
      base cm-gnmi;
      base cm-polled;
    }
    identity restconf-get-polling {
      base cm-restconf;
      base cm-polled;
    }
    identity netconf-get-polling {
      base cm-netconf;
      base cm-polled;
    }
    identity restconf-yang-push-subscription {
      base cm-restconf;
    }
    identity netconf-yang-push-subscription {
```

```
      base cm-netconf;
    }
    identity redfish-polling {
      base cm-redfish;
    }
  }
  <CODE ENDS>
```

5.2. Provider interface module for Philatelist

```
  <CODE BEGINS>
  module ietf-poweff-provider {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-provider";
    prefix ietf-poweff-provider;

    import ietf-poweff-types {
      prefix ietf-poweff-types;
    }

    organization
      "IETF OPSA (Operations and Management Area) Working Group";
    contact
      "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
       WG List:  <mailto:opsawg@ietf.org>
       Editor:  Jan Lindblad
                <mailto:jlindbla@cisco.com>
       Editor:  Snezana Mitrovic
                <mailto:snmitrov@cisco.com>
       Editor:  Marisol Palmero
                <mailto:mpalmero@cisco.com>";
    description
      "This YANG module defines the POWEFF Provider.

       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
       (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 2023-10-12 {
       description
         "Initial revision of POWEFF Provider";
       reference
         "RFC XXXX: ...";
     }

     grouping provider-g {
       container sensor-catalog {
         config false;
         container sensors {
           list sensor {
             key path;
             leaf path { type ietf-poweff-types:xpath; }
             leaf sensor-type { type identityref { base ietf-poweff-types:sensor-
type; }}

             leaf sensor-location {
               type ietf-poweff-types:something;
               description
                 "Indicates the current location where the sensor is located
                   in the chassis,typically refers to slot";
             }
             leaf sensor-state { // FIXME: What does this mean?
               type ietf-poweff-types:something;
               description
                 "Current state of the sensor";
             }
             leaf sensor-current-reading { // FIXME: Do we want a copy of the val
ue here?
               type ietf-poweff-types:something;
               description
                 "Current reading of the sensor";
             }
             leaf sensor-precision {
               type string;
               description
                 "Maximum deviation to be considered. This attribute mainly
                 will apply to drawn power, which corresponds to PSU PowerIn
                 measured power or calculated power; assuming discrepancy
                 between Real Power, power collected from a power meter, and
                 power measured or calculated from the metrics provided by
                 the sensors";
             }
             container sensor-thresholds { // FIXME: Is this for generating alarm
s, or what?
               description
                 "Threshold values for the particular sensor.
                 Default values shall beprovided as part of static data
                 but when configurable need to be pulledfrom the device.
                 Ideally, the sensor should allow configuing
```

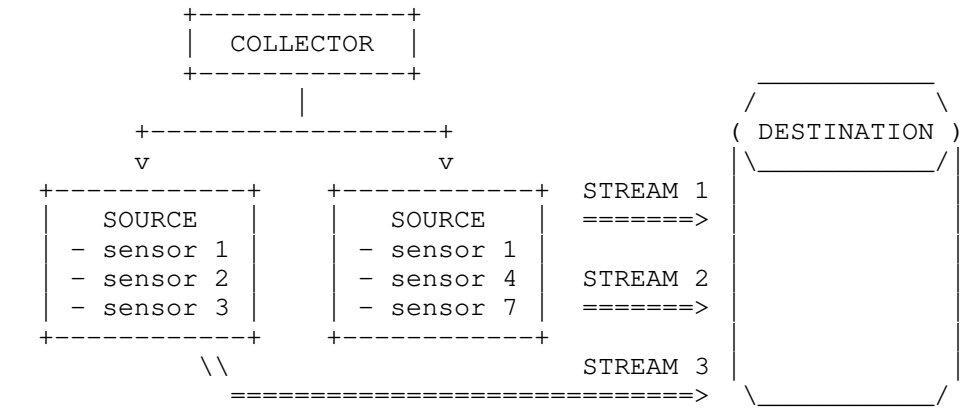
```
                thesethreshold values";

              leaf minor-low {
                type string;
                description
                  "minor-low";
              }
              leaf minor-high {
                type string;
                description
                  "minor-high";
              }
              leaf major-low {
                type string;
                description
                  "major-low";
              }
              leaf major-high {
                type string;
                description
                  "major-high";
              }
              leaf critical-low {
                type string;
                description
                  "critical-low";
              }
              leaf critical-high {
                type string;
                description
                  "critical-high";
              }
              leaf shutdown { // FIXME: What does this mean for a sensor?
                type string;
                description
                  "shutdown";
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

5.3. Collector interface module for Philatelist

```
<CODE BEGINS>
module ietf-poweff-collector {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-collector";
  prefix ietf-poweff-collector;

  import ietf-poweff-types {
    prefix ietf-poweff-types;
  }

  organization
    "IETF OPSA (Operations and Management Area) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>
     Editor:  Jan Lindblad
              <mailto:jlindbla@cisco.com>
     Editor:  Snezana Mitrovic
              <mailto:snmitrov@cisco.com>
     Editor:  Marisol Palmero
              <mailto:mpalmero@cisco.com>";
  description
    "This YANG module defines the POWEFF Collector.

     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
     (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 2023-10-12 {
    description
      "Initial revision of POWEFF Collector";
    reference
      "RFC XXXX: ...";
  }

/*
```

A COLLECTOR programs one or more SOURCE(s) to generate a STREAM of telemetry data. The STREAM is sent to a specific DESTINATION.

Each STREAM consists of timestamped sensor values from each sensor in a sensor group.




```
*/

  grouping data-endpoint-g {
    leaf url { type ietf-poweff-types:something; }
    leaf organization { type ietf-poweff-types:something; }
    leaf bucket { type ietf-poweff-types:something; }
    container impl-specific {
      list binding {
        key key;
        leaf key { type string; }
        choice value-type {
          leaf value { type string; }
          leaf-list values { type string; ordered-by user; }
          leaf env-var { type string; }
        }
      }
    }
  }

  grouping sensor-group-g {
    leaf method {
      type identityref {
        base ietf-poweff-types:collection-method;
      }
    }
    container get-static-url-once {
```

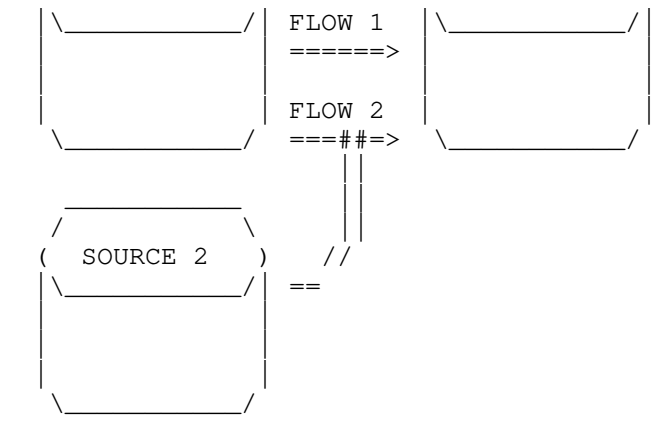
```
          when "derived-from-or-self(../method, 'ietf-poweff-types:get-static-url-
once')";
          leaf url { type ietf-poweff-types:something; }
          leaf format { type ietf-poweff-types:something; } // JSON-IETF, XML, etc
        }
        container gnmi-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:gnmi-polling')"
;
          leaf encoding { type ietf-poweff-types:something; } // self-describing-g
pb
          leaf protocol { type ietf-poweff-types:something; } // protocol grpc no-
tls
        }
        container restconf-get-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:restconf-get-po
lling')";
          leaf xxx { type string; }
        }
        container netconf-get-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:netconf-get-pol
ling')";
          leaf xxx { type string; }
        }
        container restconf-yang-push-subscription {
          when "derived-from-or-self(../method, 'ietf-poweff-types:restconf-yang-p
ush-subscription')";
          leaf xxx { type string; }
        }
        container netconf-yang-push-subscription {
          when "derived-from-or-self(../method, 'ietf-poweff-types:netconf-yang-pu
sh-subscription')";
          leaf xxx { type string; }
        }
        container redfish-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:redfish-polling
')";
          leaf xxx { type string; }
        }
        leaf frequency {
          when "derived-from(../method, 'ietf-poweff-types:cm-polled')";
          type ietf-poweff-types:sample-frequency;
        }
        list path {
          key path;
          leaf path { type ietf-poweff-types:xpath; }
          leaf sensor-type { type identityref { base ietf-poweff-types:sensor-type
; }}
          leaf attribution { type string; }
        }
      }

      grouping collector-g {
        container poweff-collector {
          container destinations {
            list destination {
              key id;
              leaf id { type ietf-poweff-types:something; }
              uses data-endpoint-g;
```

```
          }
        }

        container sensor-groups {
          list sensor-group {
            key id;
            leaf id { type ietf-poweff-types:something; }
            uses sensor-group-g;
          }
        }

        container streams {
          list stream {
            key id;
            leaf id { type ietf-poweff-types:something; }
            leaf-list source { type string; } // Implementation specific meaning
, possibly wildcards
            list sensor-group {
              key name;
              leaf name { type leafref { path ../../../sensor-groups/sensor-grou
p/id; }}
            }
            leaf destination { type leafref { path ../../../destinations/destina
tion/id; }}
          }
        }
      }
    }
  }
  <CODE ENDS>
```

5.4. Aggregator interface module for Philatelist

```
  <CODE BEGINS>
  module ietf-poweff-aggregator {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-aggregator";
    prefix ietf-poweff-aggregator;

    import ietf-poweff-types {
      prefix ietf-poweff-types;
    }
    import ietf-poweff-collector {
      prefix ietf-poweff-collector;
    }

    organization
      "IETF OPSA (Operations and Management Area) Working Group";
    contact
      "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
       WG List:  <mailto:opsawg@ietf.org>
```

```
        Editor:  Jan Lindblad
                 <mailto:jlindbla@cisco.com>
        Editor:  Snezana Mitrovic
                 <mailto:snmitrov@cisco.com>
        Editor:  Marisol Palmero
                 <mailto:mpalmero@cisco.com>";
    description
      "This YANG module defines the POWEFF Aggregator.

       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
       (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 2023-10-12 {
      description
        "Initial revision of POWEFF Aggregator";
      reference
        "RFC XXXX: ...";
    }

  /*

    An AGGREGATOR ensures data from one or more SOURCE(s) are
    combined into a FLOW using a (sequence of) OPERATIONs (OPs)
    to generate a new data set in the DESTINATION (which could
    be a new collection in the same data storage system as the
    SOURCE).

                  +-------------+
                  | AGGREGATOR  |
                  +-------------+
                         |
              +----------+----------+
              v                     v
       ____________           ____________
      /            \         /            \
     (  SOURCE 1    )       ( DESTINATION )
```




```
   */

     grouping aggregator-g {
       container poweff-aggregator {
         container sources {
           list source {
             key id;
             leaf id { type ietf-poweff-types:something; }
             uses ietf-poweff-collector:data-endpoint-g;
           }
         }
         container destinations {
           list destination {
             key id;
             leaf id { type ietf-poweff-types:something; }
             uses ietf-poweff-collector:data-endpoint-g;
           }
         }
         container flows {
           list flow {
             key id;
             leaf id { type string; }
             choice chain-position {
               container input {
                 leaf source { type leafref { path ../../../../../sources/source/
id; }}
               }
               container output {
                 leaf destination { type leafref { path ../../../../../destinatio
ns/destination/id; }}
               }
               container middle {
                 leaf-list inputs { type leafref { path ../../../../flows/flow/id
; }}
                 leaf pre-process-inputs { type leafref { path ../../../../operat
ions/operation/id; }}
```

```
                leaf aggregate { type leafref { path ../../../../operations/oper
ation/id; }}
                leaf post-process-output { type leafref { path ../../../../opera
tions/operation/id; }}
              }
            }
          }
        }
        container operations {
          list operation {
            key id;
            leaf id { type ietf-poweff-types:something; }
            choice op-type {
              container linear-sum {}
              container linear-average {}
              container linear-max {}
              container linear-min {}
              container rolling-average {
                leaf timespan { type ietf-poweff-types:something; }
              }
              container filter-age {
                leaf min-age { type ietf-poweff-types:something; }
                leaf max-age { type ietf-poweff-types:something; }
              }
              container function {
                leaf name { type ietf-poweff-types:something; }
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Security Considerations

TODO Security

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[I-D.draft-kll-yang-label-tsdb-00]
Larsson, K., "Mapping YANG Data to Label-Set Time Series",
Work in Progress, Internet-Draft, draft-kll-yang-label-

tsdb-00, 18 October 2023,
<https://datatracker.ietf.org/doc/html/draft-kll-yang-label-tsdb-00>.

[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>.

[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>.

[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/rfc/rfc8174>.

8.2. Informative References

[I-D.draft-claise-netconf-metadata-for-collection-03]
Claise, B., Nayyar, M., and A. R. Sesani, "Per-Node Capabilities for Optimum Operational Data Collection", Work in Progress, Internet-Draft, draft-claise-netconf-metadata-for-collection-03, 25 January 2022, <https://datatracker.ietf.org/doc/html/draft-claise-netconf-metadata-for-collection-03>.

[I-D.draft-ietf-opsawg-collected-data-manifest-01]
Claise, B., Quilbeuf, J., Lopez, D., Martinez-Casanueva, I. D., and T. Graf, "A Data Manifest for Contextualized Telemetry Data", Work in Progress, Internet-Draft, draft-ietf-opsawg-collected-data-manifest-01, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-collected-data-manifest-01>.

Acknowledgments

Kristian Larsson has provided invaluable insights, experience and validation of the design. Many thanks to the entire POWEFF team for their committment, flexibility and hard work behind this. Hat off to Benoît Claise, who inspires by the extensive work produced in IETF over the years, and in this area in particular.

Author’s Address

Jan Lindblad
Cisco
Email: jlindbla@cisco.com

Operations and Management Area Working Group
Internet-Draft
Intended status: Informational
Expires: 2 September 2024

D. Lopez
A. Pastor
Telefonica
A. Huang Feng
INSA-Lyon
H. Birkholz
Fraunhofer SIT
1 March 2024

Applying COSE Signatures for YANG Data Provenance
draft-lopez-opsawg-yang-provenance-02

Abstract

This document defines a mechanism based on COSE signatures to provide and verify the provenance of YANG data, so it is possible to verify the origin and integrity of a dataset, even when those data are going to be processed and/or applied in workflows where a crypto-enabled data transport directly from the original data stream is not available. As the application of evidence-based OAM automation and the use of tools such as AI/ML grow, provenance validation becomes more relevant in all scenarios. The use of compact signatures facilitates the inclusion of provenance strings in any YANG schema requiring them.

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://dr2lopez.github.io/yang-provenance/draft-lopez-opsawg-yang-provenance.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-lopez-opsawg-yang-provenance/.

Discussion of this document takes place on the Operations and Management Area Working Group Working Group mailing list (mailto:opsawg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/opsawg/. Subscribe at https://www.ietf.org/mailman/listinfo/opsawg/.

Source for this draft and an issue tracker can be found at https://github.com/dr2lopez/yang-provenance.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
2. Conventions and Definitions 4
3. Defining Provenance Elements 4
3.1. Provenance Signature Strings 5
3.2. Signature and Verification Procedures 5
3.3. Canonicalization 6
3.4. Provenance-Signature YANG Module 6
4. Enclosing Methods 7
4.1. Including a Provenance Leaf in a YANG Element 8
4.2. Including a Provenance Signature in NETCONF Event Notifications and YANG-Push Notifications 10
4.2.1. YANG Tree Diagram 11
4.2.2. YANG Module 11
4.3. Including Provenance as Metadata in YANG Instance Data . 13
4.3.1. YANG Module 13

4.4. Inclduing Provenance in YANG Annotations 13
4.4.1. YANG Module 14
5. Security Considerations 14
6. IANA Considerations 14
6.1. IETF XML Registry 14
6.2. YANG Module Name 15
7. References 15
7.1. Normative References 15
7.2. Informative References 17
Acknowledgments 17
Authors' Addresses 17

1. Introduction

OAM automation, generally based on closed-loop principles, requires at least two datasets to be used. Using the common terms in Control Theory, we need those from the plant (the network device or segment under control) and those to be used as reference (the desired values of the relevant data). The usual automation behavior compares these values and takes a decision, by whatever the method (algorithmic, rule-based, an AI model tuned by ML...) to decide on a control action according to this comparison. Assurance of the origin and integrity of these datasets, what we refer in this document as "provenance", becomes essential to guarantee a proper behavior of closed-loop automation.

When datasets are made available as an online data flow, provenance can be assessed by properties of the data transport protocol, as long as some kind of cryptographic protocol is used for source authentication, with TLS, SSH and IPsec as the main examples. But when these datasets are stored, go through some pre-processing or aggregation stages, or even cryptographic data transport is not available, provenance must be assessed by other means.

The original use case for this provenance mechanism is associated with [YANGmanifest], in order to provide a proof of the origin and integrity of the provided metadata, and therefore the examples in this document use the modules described there, but it soon became clear that it could be extended to any YANG datamodel to support provenance evidence. An analysis of other potential use cases suggested the interest of defining an independent, generally applicable mechanism.

Provenance verification by signatures incorporated in YANG data can be applied to any data processing pipeline, whether they rely on an online flow or use some kind of data store, such as data lakes or time-series databases. The application of recorded data for ML training or validation constitute the most relevant examples of these scenarios.

This document provides a mechanism for including digital signatures within YANG data. It applies COSE [RFC9052] to make the signature compact and reduce the resources required for calculating it. This mechanism is applicable to any serialization of the YANG data supporting a clear method for canonicalization, but this document considers three base ones: CBOR, JSON and XML.

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.

The term "data provenance" refers to a documented trail accounting for the origin of a piece of data and where it has moved from to where it is presently. The signature mechanism provided here can be recursively applied to allow this accounting for YANG data.

3. Defining Provenance Elements

The provenance for a given YANG element MUST be convened by a leaf element, containing the COSE signature bitstring built according to the procedure defined below in this section. The provenance leaf MUST be of type provenance-signature, defined as follows:

```
typedef provenance-signature {
     type binary;
     description
      "The provenance-signature type represents a digital signature
       corresponding to the associated YANG element. The signature is based
       on COSE and generated using a canonicalized version of the
       associated element.";
     reference
      "RFC 9052: CBOR Object Signing and Encryption (COSE): Structures and Proces
s
       draft-lopez-opsawg-yang-provenance";
}
```

3.1. Provenance Signature Strings

Provenance signature strings are COSE single signature messages with [nil] payload, according to COSE conventions and registries, and with the following structure (as defined by [RFC9052], Section 4.2):

```
COSE_Sign1 = [
protected /algorithm-identifier, kid, serialization-method/
unprotected /algorithm-parameters/
signature /using as external data the content of the YANG
           (meta-)data without the signature leaf/
]
```

The COSE_Sign1 procedure yields a bitstring when building the signature and expects a bitstring for checking it, hence the proposed type for provenance signature leaves. The structure of the COSE_Sign1 consists of:

* The algorithm-identifier, which MUST follow COSE conventions and registries.

* The kid (Key ID), to be locally agreed, used and interpreted by the signer and the signature validator. URIs [RFC3986] and RFC822-style [RFC5322] identifiers are typical values to be used as kid.

* The serialization-method, a string identifying the YANG serialization in use. It MUST be one of the three possible values "xml" (for XML serialization [RFC7950]), "json" (for JSON serialization [RFC7951]) or "cbor" (for CBOR serialization [RFC9254]).

* The value algorithm-parameters, which MUST follow the COSE conventions for providing relevant parameters to the signing algorithm.

* The signature for the YANG element provenance is being established for, to be produced and verified according to the procedure described below for each one of the enclosing methods for the provenance string described below.

3.2. Signature and Verification Procedures

To keep a concise signature and avoid the need for wrapping YANG constructs in COSE envelopes, the whole signature MUST be built and verified by means of externally supplied data, as defined in [RFC9052], Section 4.3, with a [nil] payload.

The byte strings to be used as input to the signature and verification procedures MUST be built by:

* Selecting the exact YANG content to be used, according to the corresponding enclosing methods.

* Applying the corresponding canonicalization method as described in the following section.

3.3. Canonicalization

Signature generation and verification require a canonicalization method to be applied, that depends on the serialization used. According to the three types of serialization defined, the following canonicalization methods MUST be applied:

* For CBOR, length-first core deterministic encoding, as defined by [RFC8949].

* For JSON, JSON Canonicalization Scheme (JCS), as defined by [RFC8785].

* For XML, Exclusive XML Canonicalization 1.0, as defined by [XMLSig].

3.4. Provenance-Signature YANG Module

This module defines a provenance-signature type to be used in other YANG modules.

```
<CODE BEGINS> file "ietf-yang-provenance@2024-02-28.yang"
module ietf-yang-provenance {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-provenance";
  prefix iyangprov;

  organization "IETF OPSAWG (Operations and Management Area Working Group)";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>

     Authors:  Alex Huang Feng
               <mailto:alex.huang-feng@insa-lyon.fr>
               Diego Lopez
               <mailto:diego.r.lopez@telefonica.com>
               Antonio Pastor
               <mailto:antonio.pastorperales@telefonica.com>
```

```
                Henk Birkholz
                <mailto:henk.birkholz@sit.fraunhofer.de>";

    description
      "Defines a binary provenance-signature type to be used in other YANG
      modules.

      Copyright (c) 2024 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; see the RFC
      itself for full legal notices.";

    revision 2024-02-28 {
      description
        "First revision";
      reference
        "RFC XXXX: Applying COSE Signatures for YANG Data Provenance";
    }

    typedef provenance-signature {
      type binary;
      description
        "The provenance-signature type represents a digital signature
        corresponding to the associated YANG element. The signature is based
        on COSE and generated using a canonicalized version of the
        associated element.";
      reference
        "RFC XXXX: Applying COSE Signatures for YANG Data Provenance";
    }
  }
  <CODE ENDS>
```

4. Enclosing Methods

Once defined the procedures for generating and verifying the provenance signature string, let's consider how these signatures can be integrated with the associated YANG data by enclosing the signature in the data structure. This document considers four different enclosing methods, suitable for different stages of the YANG schema and usage patterns of the YANG data. The enclosing method defines not only how the provenance signature string is

combined with the signed YANG data but also the specific procedure for selecting the specific YANG content to be processed when signing and verifying

4.1. Including a Provenance Leaf in a YANG Element

This enclosing method requires a specific element in the YANG schema defining the element to be signed (the enclosing element), and thus implies considering provenance signatures when creating the corresponding YANG module, or the update of existing modules willing to support this provenance enclosing method.

When using this enclosing method, a provenance-signature leaf MAY appear at any position in the enclosing element, but only one such leaf MUST be defined for the enclosing element. If the enclosing element contains other non-leaf elements, they MAY provide their own provenance-signature leaf, according to the same rule. In this case, the provenance-signature leaves in the children elements are applicable to the specific child element where they are enclosed, while the provenance-signature leaf enclosed in the top-most element is applicable to the whole element contents, including the children provenance-signature leaf themselves. This allows for recursive provenance validation, data aggregation, and the application of provenance verification of relevant children elements at different stages of any data processing pipeline.

The specific YANG content to be processed SHALL be generated by taking the whole enclosing element and eliminiating the leaf containing the provenance signature string.

As example, let us consider the two modules proposed in [YANGmanifest]. For the platform-manifest module, the provenance for a platform would be provided by the optional platform-provenance leaf shown below:

```
module: ietf-platform-manifest
  +--ro platforms
     +--ro platform* [id]
       +--ro id                       string
       +--ro name?                    string
       +--ro vendor?                  string
       +--ro vendor-pen?              uint32
       +--ro software-version?        string
       +--ro software-flavor?         string
       +--ro os-version?              string
       +--ro os-type?                 string
       +--ro platform-provenance?     provenance-signature
       +--ro yang-push-streams
       |  +--ro stream* [name]
       |     +--ro name
       |     +--ro description?
       +--ro yang-library
       + . . .
       .
       .
       .
```

For data collections, the provenance of each one would be provided by the optional collector-provenance leaf, as shown below:

```
module: ietf-data-collection-manifest
  +--ro data-collections
     +--ro data-collection* [platform-id]
     +--ro platform-id
     |       -> /p-mf:platforms/platform/id
     +--ro collector-provenance?   provenance-signature
     +--ro yang-push-subscriptions
       +--ro subscription* [id]
         +--ro id
         |      sn:subscription-id
         +
         .
         .
         .
     + . . .
     |
     .
     .
     .
```

Note how, in the two examples, the element bearing the provenance signature appears at different positions in the enclosing element. And note that, for processing the element for signature generation

and verification, the signature element MUST be eliminated from the enclosing element before applying the corresponding canonicalization method.

Note that, in application of the recursion mechanism described above, a provenance element could be included at the top of any of the collections, supporting the verification of the provenance of the collection itself (as provided by a specific collector), without interfering with the verification of the provenance of each of the collection elements. As an example, in the case of the platform manifests it would look like:

```
module: ietf-platform-manifest
  +--ro platforms
     +--ro platform-collection-provenance? provenance-signature
     +--ro platform* [id]
       +--ro platform-provenance?          provenance-signature
       +--ro id                            string
       +--ro name?                         string
       +--ro vendor?                       string
       + . . .
       .
       .
       .
```

Note here that, to generate the YANG content to be processed in the case of the collection the provenance leafs of the indivual elements SHALL NOT be eliminated, as it SHALL be the case when generating the YANG content to be processed for each individual element in the collection.

4.2. Including a Provenance Signature in NETCONF Event Notifications and YANG-Push Notifications

The signature mechanism proposed in this document MAY be used with NETCONF Event Notifications [RFC5277] and YANG-Push [RFC8641] to sign the generated notifications directly from the publisher nodes. The signature is added to the header of the Notification along with the eventTime leaf.

The YANG content to be processed MUST consist of the content of the notificationContent element.

The following sections define the YANG module augmenting the ietf-notification module.

4.2.1. YANG Tree Diagram

The following is the YANG tree diagram [RFC8340] for the ietf-notification-provenance augmentation within the ietf-notification.

```
module: ietf-notification-provenance

  augment-structure /inotif:notification:
    +-- notification-provenance?   iyangprov:provenance-signature
```

And the following is the full YANG tree diagram for the notification.

```
module: ietf-notification

  structure notification:
    +-- eventTime                                yang:date-and-time
    +-- inotifprov:notification-provenance?   iyangprov:provenance-signature
```

4.2.2. YANG Module

The module augments ietf-notification module [I-D.ahuang-netconf-notif-yang] adding the signature leaf in the notification header.

```
<CODE BEGINS> file "ietf-notification-provenance@2024-02-28.yang"
module ietf-notification-provenance {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-notification-provenance";
  prefix inotifprov;

  import ietf-notification {
    prefix inotif;
    reference
      "draft-ahuang-netconf-notif-yang: NETCONF Event Notification YANG";
  }
  import ietf-yang-provenance {
    prefix iyangprov;
    reference
      "RFC XXXX: Applying COSE Signatures for YANG Data Provenance";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions";
  }

  organization "IETF OPSAWG (Operations and Management Area Working Group)";
```

```
    contact
      "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
       WG List:  <mailto:opsawg@ietf.org>

       Authors:  Alex Huang Feng
                 <mailto:alex.huang-feng@insa-lyon.fr>
                 Diego Lopez
                 <mailto:diego.r.lopez@telefonica.com>
                 Antonio Pastor
                 <mailto:antonio.pastorperales@telefonica.com>
                 Henk Birkholz
                 <mailto:henk.birkholz@sit.fraunhofer.de>";

    description
      "Defines a binary provenance-signature type to be used in other YANG
      modules.

      Copyright (c) 2024 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; see the RFC
      itself for full legal notices.";

    revision 2024-02-28 {
      description
        "First revision";
      reference
        "RFC XXXX: Applying COSE Signatures for YANG Data Provenance";
    }

    sx:augment-structure "/inotif:notification" {
      leaf notification-provenance {
        type iyangprov:provenance-signature;
        description
          "COSE signature of the content of the Notification for
          provenance verification.";
      }
    }
  }
  <CODE ENDS>
```

4.3. Including Provenance as Metadata in YANG Instance Data

Provenance signature strings can be included as part of the metadata in YANG instance data files, as defined in [RFC9195] for data at rest. The augmented YANG tree diagram including the provenance signature is as follows:

```
module: ietf-yang-instance-data-provenance
  augment-structure instance-data-set:
    +--provenance-string?   provenance-signature
```

The provenance signature string in this enclosing method applies to whole content-data element in instance-data-set, independently of whether those data contain other provenance signature strings by applying other enclosing methods.

The specific YANG content to be processed SHALL be generated by taking the contents of the content-data element and applying the corresponding canonicalization method.

TBD: Example of YANG data file with provenace strings, probably using the same examples of [RFC9195].

4.3.1. YANG Module

TBD: YANG module derived from [RFC9195], named "ietf-yang-instance-data-provenance"

4.4. Inclduing Provenance in YANG Annotations

The use of annotations as defined in [RFC7952] seems a natural enclosing method, dealing with the provenance signature string as metadata and not requiring modification of existing YANG schemas.The provenance-string annotation is defined as follows:

```
 md:annotation provenance-string {
       type provenance-signature;
       description
         "This annotation contains a digital signature corresponding
          to the YANG element in which it appears.";
     }
```

The specific YANG content to be processed SHALL be generated by eliminating the provenance-string (encoded according to what is described in Section 5 of [RFC7952]) from the element it applies to, before invoking the corresponding canonicalization method. In application of the general recursion principle for provenance signature strings, any other provenance strings within the element to which the provenance-string applies SHALL be left as they appear, whatever the enclosing method used for them.

TBD: Provide an example for a provenance-string annotation, possibly follwing the examples in [RFC7952].

4.4.1. YANG Module

TBD: YANG module based on [RFC7952], named "yang-provenance-metadata"

5. Security Considerations

The provenance assessment mechanism described in this document relies on COSE [RFC9052] and the deterministic encoding or canonicalization procedures described by [RFC8949], [RFC8785] and [XMLSig]. The security considerations made in these references are fully applicable here.

The verification step depends on the association of the kid (Key ID) with the proper public key. This is a local matter for the verifier and its specification is out of the scope of this document. The use of certificates, PKI mechanisms, or any other secure distribution of id-public key mappings is RECOMMENDED.

6. IANA Considerations

6.1. IETF XML Registry

This document registers the following URIs in the "IETF XML Registry" [RFC3688]:

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

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

6.2. YANG Module Name

This document registers the following YANG modules in the "YANG Module Names" registry [RFC6020]:

```
  name: ietf-yang-provenance
  namespace: urn:ietf:params:xml:ns:yang:ietf-yang-provenance
  prefix: iyangprov
  reference: RFC XXXX

  name: ietf-notification-provenance
  namespace: urn:ietf:params:xml:ns:yang:ietf-notification-provenance
  prefix: inotifprov
  reference: RFC XXXX
```

TBD: Others? At least for the two additional enclosing methods (instance files and annotations)

7. References

7.1. Normative References

[I-D.ahuang-netconf-notif-yang]
Feng, A. H., Francois, P., Graf, T., and B. Claise, "YANG model for NETCONF Event Notifications", Work in Progress, Internet-Draft, draft-ahuang-netconf-notif-yang-04, 21 January 2024, <https://datatracker.ietf.org/doc/html/draft-ahuang-netconf-notif-yang-04>.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/rfc/rfc5277>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/rfc/rfc5322>.

[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>.

[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>.

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

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

[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/rfc/rfc8174>.

[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>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/rfc/rfc8641>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9195] Lengyel, B. and B. Claise, "A File Format for YANG Instance Data", RFC 9195, DOI 10.17487/RFC9195, February 2022, <https://www.rfc-editor.org/rfc/rfc9195>.

[RFC9254] Veillette, M., Ed., Petrov, I., Ed., Pelov, A., Bormann, C., and M. Richardson, "Encoding of Data Modeled with YANG in the Concise Binary Object Representation (CBOR)", RFC 9254, DOI 10.17487/RFC9254, July 2022, <https://www.rfc-editor.org/rfc/rfc9254>.

[XMLSig] "XML Signature Syntax and Processing Version 2.0", n.d., <https://www.w3.org/TR/xmldsig-core2/>.

7.2. Informative References

[YANGmanifest] Claise, B., Quilbeuf, J., Lopez, D., Martinez-Casanueva, I. D., and T. Graf, "A Data Manifest for Contextualized Telemetry Data", Work in Progress, Internet-Draft, draft-ietf-opsawg-collected-data-manifest-02, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-collected-data-manifest-02>.

Acknowledgments

This document is based on work partially funded by the EU H2020 project SPIRS (grant 952622), and the EU Horizon Europe projects PRIVATEER (grant 101096110), HORSE (grant 101096342) and ACROSS (grant 101097122).

Authors' Addresses

Diego Lopez
Telefonica
Email: diego.r.lopez@telefonica.com

Antonio Pastor
Telefonica
Email: antonio.pastorperales@telefonica.com

Alex Huang Feng
INSA-Lyon
Email: alex.huang-feng@insa-lyon.fr

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2024

Q. Ma, Ed.
Q. Wu
Huawei
M. Boucadair, Ed.
Orange
D. King
Lancaster University

1 March 2024

A Common YANG Data Model for Scheduling
draft-ma-opsawg-schedule-yang-04

Abstract

This document defines a common schedule YANG module which is designed to be applicable for scheduling information such as event, policy, services, or resources based on date and time. For the sake of better modularity, the module includes basic, intermediate, and advanced versions of recurrence related groupings.

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 2 September 2024.

Table of Contents

1. Introduction 2
1.1. Editorial Note (To be removed by RFC Editor) 3
2. Conventions and Definitions 3
3. Module Overview 4
3.1. The "generic-schedule-params" Grouping 4
3.2. The "period-of-time" Grouping 5
3.3. The "recurrence" Grouping 6
3.4. The "recurrence-with-date-times" Grouping 8
3.5. The "icalendar-recurrence" Grouping 9
3.6. The "schedule-status" Grouping 11
4. Features and Augmentations 12
5. Note and Restrictions 13
6. The "ietf-schedule" YANG Module 13
7. Security Considerations 28
8. IANA Considerations 28
8.1. The "IETF XML" Registry 29
8.2. The "YANG Module Names" Registry 29
9. References 29
9.1. Normative References 29
9.2. Informative References 30
Appendix A. Examples of Format Representation 32
A.1. The "period-of-time" Grouping 32
A.2. The "recurrence" Grouping 32
A.3. The "recurrence-with-date-times" Grouping 33
A.4. The "icalendar-recurrence" Grouping 34
Appendix B. Examples of Using/Extending the "ietf-schedule" Module 35
B.1. Example: Schedule Tasks to Execute Based on a Recurrence Rule 35
B.2. Example: Schedule Network Properties to Change Based on Date and Time 38
Acknowledgments 41
Authors' Addresses 41

1. Introduction

Several specifications include a provision for scheduling. Examples of such specifications are (but not limited to) [I-D.ietf-opsawg-ucl-acl], [I-D.contreras-opsawg-scheduling-oam-tests], and [I-D.united-tvr-schedule-yang]. Both [I-D.ietf-opsawg-ucl-acl] and [I-D.contreras-opsawg-scheduling-oam-tests] use the "ietf-schedule"

module initially specified in [I-D.ietf-opsawg-ucl-acl].

Given that the applicability of the "ietf-schedule" module is more general than scheduled policy and OAM tests, this document defines "ietf-schedule" as a common schedule YANG module. The module includes a set of reusable groupings which are designed to be applicable for scheduling information such as event, policy, services or resources based on date and time.

Examples to illustrate the use of the common groupings are provided in Appendix A. Also, sample modules to exemplify how future modules can use the extensibility provisions in "ietf-schedule" are provided in Appendix B.

1.1. Editorial Note (To be removed by RFC Editor)

Note to the RFC Editor: This section is to be removed prior to publication.

This document contains placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Please apply the following replacements:

* XXXX --> the assigned RFC number for this draft

* YYYY --> the assigned RFC number for [I-D.ietf-netmod-rfc6991-bis]

* 2023-01-19 --> the actual date of the publication of this document

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.

The meanings of the symbols in tree diagrams are defined in [RFC8340].

Also, this document uses the YANG terminology defined in Section 3 of [RFC7950].

3. Module Overview

The "ietf-schedule" module (Section 6) defines the following groupings:

* "generic-schedule-params" (Section 3.1)

* "period-of-time" (Section 3.2)

* "recurrence" (Section 3.3)

* "recurrence-with-date-times" (Section 3.4)

* "icalendar-recurrence" (Section 3.5)

* "schedule-status" (Section 3.6)

Figure 1 provides an overview of the tree structure [RFC8340] of the "ietf-schedule" module in terms of its groupings.

```
module: ietf-schedule

  grouping generic-schedule-params:
    ...
  grouping period-of-time:
    ...
  grouping recurrence:
    ...
  grouping recurrence-with-date-times:
    ...
  grouping icalendar-recurrence:
    ...
  grouping schedule-status:
    ...
```

Figure 1: Overall Schedule Tree Structure

Each of these groupings is presented in the following subsections. Examples are provided in Appendix A.

3.1. The "generic-schedule-params" Grouping

The "generic-schedule-params" grouping (Figure 2) specifies a set of configuration parameters that are used by a system for validating requested schedules. These parameters apply to all schedules on a system and are meant to provide guards against stale configuration, too short schedule requests that would prevent validation by admins of some critical systems, etc.

```
module: ietf-schedule

  grouping generic-schedule-params:
    +-- time-zone-identifier?   sys:timezone-name
    +-- validity?               yang:date-and-time
    +-- max-allowed-start?      yang:date-and-time
    +-- min-allowed-start?      yang:date-and-time
    +-- max-allowed-end?        yang:date-and-time
    +-- discard-action?         enumeration
  grouping period-of-time:
    ...
  grouping recurrence:
    ...
  grouping recurrence-with-date-times:
    ...
  grouping icalendar-recurrence:
    ...
  grouping schedule-status:
    ...
```

Figure 2: Generic Schedule Configuration Tree Structure

3.2. The "period-of-time" Grouping

The "period-of-time" grouping (Figure 3) represents a time period using either a start ("period-start") and end date and time ("period-end"), or a start ("period-start") and a positive time duration ("duration"). For the first format, the start of the period MUST be before the end of the period.

```
module: ietf-schedule

  grouping generic-schedule-params:
    ...
  grouping period-of-time:
    +-- period-start?           yang:date-and-time
    +-- time-zone-identifier?   sys:timezone-name
    +-- (period-type)?
       +--:(explicit)
       |  +-- period-end?       yang:date-and-time
       +--:(duration)
          +-- duration?         duration
  grouping recurrence:
    ...
  grouping recurrence-with-date-times:
    ...
  grouping icalendar-recurrence:
    ...
  grouping schedule-status:
    ...
```

Figure 3: Period of Time Grouping Tree Structure

3.3. The "recurrence" Grouping

The "recurrence" grouping (Figure 4) specifies a simple recurrence rule, the definition conforms to part of the "recurrence rule" properties in Section 3.3.10 of [RFC5545].

```
module: ietf-schedule

  grouping generic-schedule-params:
    ...
  grouping period-of-time:
    ...
  grouping recurrence:
    +-- recurrence-first
    |  +-- date-time-start?        union
    |  +-- time-zone-identifier?   sys:timezone-name
    |  +-- duration?               duration
    +-- frequency?          identityref
    +-- interval?           uint32
    +-- (recurrence-bound)?
       +--:(until)
       |  +-- until?        union
       +--:(count)
          +-- count?        uint32
  grouping recurrence-with-date-times:
    ...
  grouping icalendar-recurrence:
    ...
  grouping schedule-status:
    ...
```

Figure 4: Recurrence Grouping Tree Structure

The "recurrence-first" container defines the first instance in the recurrence set. It also determines the start time and duration (if specified) of subsequent recurrence instances. If the "date-time-start" node is specified as a date-no-zone value type with no duration specified, the recurrence's duration is taken to be one day.

The frequency ("frequency") identifies the type of recurrence rule. For example, a "daily" frequency value specifies repeating events based on an interval of a day or more.

The interval represents at which intervals the recurrence rule repeats. For example, within a daily recurrence rule, an interval value of "8" means every eight days.

The repetition can be scoped by a specified end time or by a count of occurrences, indicated by the "recurrence-bound" choice. The "date-time-start" value always counts as the first occurrence.

3.4. The "recurrence-with-date-times" Grouping

The "recurrence-with-date-times" grouping (Figure 5) uses the "recurrence" grouping (Section 3.3) and adds a "date-times-choice" statement to define an aggregate set of repeating occurrences.

```
module: ietf-schedule

  grouping generic-schedule-params:
    ...
  grouping period-of-time:
    ...
  grouping recurrence:
    ...
  grouping recurrence-with-date-times:
    +-- recurrence-first
    |  +-- date-time-start?        union
    |  +-- time-zone-identifier?   sys:timezone-name
    |  +-- duration?               duration
    +-- frequency?                 identityref
    +-- interval?                  uint32
    +-- (recurrence-bound)?
    |  +--:(until)
    |  |  +-- until?               union
    |  +--:(count)
    |     +-- count?               uint32
    +-- (date-times-choice)?
       +--:(date-time)
       |  +-- date-times*          yang:date-and-time
       +--:(date)
       |  +-- dates*               yang:date-no-zone
       +--:(period-timeticks)
       |  +-- period-timeticks* [period-start]
       |     +-- period-start?   yang:timeticks
       |     +-- period-end?     yang:timeticks
       +--:(period)
          +-- period* [period-start]
             +-- period-start?           yang:date-and-time
             +-- time-zone-identifier?   sys:timezone-name
             +-- (period-type)?
                +--:(explicit)
                |  +-- period-end?       yang:date-and-time
                +--:(duration)
                   +-- duration?         duration
  grouping icalendar-recurrence:
    ...
  grouping schedule-status:
    ...
```

Figure 5: Recurrence with Date Times Grouping Tree Structure

The recurrence instances are defined by the union of occurrences defined by both date-times and recurrence rule. When duplicate instances are generated, only one recurrence is considered.

Date-times definition inside "recurrence-with-date-times" grouping refers to but does not fully comply with Section 3.8.5.2 of [RFC5545]. A timeticks type based period is added.

3.5. The "icalendar-recurrence" Grouping

The "icalendar-recurrence" grouping (Figure 6) uses the "recurrence-with-date-times" grouping (Section 3.4) and add more data nodes to enrich the definition of recurrence. The structure of the "icalendar-recurrence" grouping conforms to the definition of recurrence component defined in Section 3.8.5 of [RFC5545].

```
module: ietf-schedule

  grouping generic-schedule-params:
    ...
  grouping period-of-time:
    ...
  grouping recurrence:
    ...
  grouping recurrence-with-date-times:
    ...
  grouping icalendar-recurrence:
    +-- recurrence-first
    |  +-- date-time-start?        union
    |  +-- time-zone-identifier?   sys:timezone-name
    |  +-- duration?               duration
    +-- frequency?                 identityref
    +-- interval?                  uint32
    +-- (recurrence-bound)?
    |  +--:(until)
    |  |  +-- until?               union
    |  +--:(count)
    |     +-- count?               uint32
    +-- (date-times-choice)?
    |  +--:(date-time)
    |  |  +-- date-times*          yang:date-and-time
    |  +--:(date)
    |  |  +-- dates*               yang:date-no-zone
    |  +--:(period-timeticks)
    |  |  +-- period-timeticks* [period-start]
    |  |     +-- period-start?    yang:timeticks
```

```
      |  |     +-- period-end?     yang:timeticks
      |  +--:(period)
      |     +-- period* [period-start]
      |        +-- period-start?           yang:date-and-time
      |        +-- time-zone-identifier?   sys:timezone-name
      |        +-- (period-type)?
      |           +--:(explicit)
      |           |  +-- period-end?       yang:date-and-time
      |           +--:(duration)
      |              +-- duration?         duration
      +-- bysecond*                  uint32
      +-- byminute*                  uint32
      +-- byhour*                    uint32
      +-- byday* [weekday]
      |  +-- direction*   int32
      |  +-- weekday?     schedule:weekday
      +-- bymonthday*                int32
      +-- byyearday*                 int32
      +-- byyearweek*                int32
      +-- byyearmonth*               uint32
      +-- bysetpos*                  int32
      +-- workweek-start?            schedule:weekday
      +-- exception-dates*           union
    grouping schedule-status:
      ...
```

Figure 6: iCalendar Recurrence Grouping Tree Structure

An array of the "bysecond" (or "byminut", "byhour") specifies a list of seconds within a minute (or minutes within an hour, hours of the day).

The parameter "byday" specifies a list of days of the week, with an optional direction which indicates the nth occurrence of a specific day within the "monthly" or "yearly" frequency. For example, within a "monthly" rule, the "weekday" with a value of "monday" and the "direction" with a value of "-1" represents the last Monday of the month.

An array of the "bymonthday" (or byyearday", "byyearweek", or "byyearmonth") specifies a list of days of the month (or days of the year, weeks of the year, or months of the year).

The "bysetpos" conveys a list of values that corresponds to the nth occurrence within the set of recurrence instances to be specified. For example, in a "monthly" recurrence rule, the "byday" data node specifies every Monday of the week, the "bysetpos" with value of "-1" represents the last Monday of the month. Not setting the "bysetpos" data node represents every Monday of the month.

The "workweek-start" data node specifies the day on which the week starts. This is significant when a "weekly" recurrence rule has an interval greater than 1, and a "byday" data node is specified. This is also significant when in a "yearly" rule and a "byyearweek" is specified. The default value is "monday".

The "exception-dates" data node specifies a list of exceptions for recurrence. The final recurrence set is generated by gathering all of the date and time values generated by any of the specified recurrence rule and date-times, and then excluding any start date and time values specified by "exception-dates" parameter.

3.6. The "schedule-status" Grouping

The "schedule-status" grouping (Figure 7) defines common parameters for scheduling management/status exposure.

```
module: ietf-schedule

  grouping generic-schedule-params:
    ...
  grouping period-of-time:
    ...
  grouping recurrence:
    ...
  grouping recurrence-with-date-times:
    ...
  grouping icalendar-recurrence:
    ...
  grouping schedule-status:
    +-- schedule-id?           string
    +-- state?                 identityref
    +-- version?               uint16
    +--ro schedule-type?         identityref
    +--ro last-update?           yang:date-and-time
    +--ro counter?               uint32
    +--ro last-occurrence?       yang:date-and-time
    +--ro upcoming-occurrence?   yang:date-and-time
```

Figure 7: Schedule Status Grouping Tree Structure

The "schedule-id" parameter is useful to uniquely identify a schedule in a network device or controller if multiple scheduling contexts exists.

The "state" parameter is defined to configure/expose the scheduling state, depending on the use of the grouping. The "identityref" type is used for this parameter to allow extensibility in future modules. For example, a "conflict" state is valid in scheduling contexts where multiple systems struggle for the scheduling of the same property. The conflict may be induced by, e.g., multiple entities managing the schedules for the same target component.

The "version" parameter is used to track the current schedule version information. The version can be bumped by the entity who create the schedule. The "last-update" parameter identifies when the schedule was last modified. In some contexts, this parameter can be used to track the configuration of a given schedule. In such cases, the "version" may not be used.

The "schedule-type" parameter identifies the type of the current schedule. The "counter", "last-occurrence", and "upcoming-occurrence" data nodes are only avaliable when the "schedule-type" is "recurrence".

The current grouping captures common parameters that is applicable to typical scheduling contexts known so far. Future modules can define other useful parameters as needed. For example, in a scheduling context with multiple system sources to feed the schedules, the "source" and "precedence" parameters may be needed to reflect how schedules from different sources should be prioritised.

4. Features and Augmentations

The "ietf-schedule" data model defines the recurrence related groupings using a modular approach. Basic, intermediate, and advanced representation of recurrence groupings are defined, with each reusing the previous one and adding more parameters. To allow for different options, two features are defined in the data model:

* 'basic-recurrence-supported'

* 'icalendar-recurrence-supported'

Appendix B.1 provides an example about how that could be used. Implementations may support a basic recurrence rule or an advanced one as needed, by declaring different features. Whether only one or both features are supported is implementation specific and depend on specific scheduling context.

These groupings can also be augmented to support specific needs. As an example, Appendix B.2 demonstrates how additional parameters can be added to comply with specifc schedule needs.

5. Note and Restrictions

There are some restrictions that need to be followed when using groupings defined in "ietf-schedule" yang module:

* The instant in time represented by "period-start" MUST be before the "period-end" for "period-of-time" grouping

* The combination of the day, month, and year represented for date and time value MUST be valid. See Section 5.7 of [RFC3339] for the maxinum day number based on the month and year.

* The second MUST have the value "60" at the end of months in which a leap second occurs for date and time value

* Care must be taken when defining recurrence occurring very often and frequent that can be an additional source of attacks by keeping the system permanently busy with the management of scheduling

* Schedules received with a starting time in the past with respect to current time SHOULD be ignored

6. The "ietf-schedule" YANG Module

This module imports types defined in [I-D.ietf-netmod-rfc6991-bis].

<CODE BEGINS> file "ietf-schedule@2023-01-19.yang"

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

  import ietf-yang-types {
    prefix yang;
    revision-date 2023-01-23;
    reference
      "RFC YYYY: Common YANG Data Types";
  }

  import ietf-system {
    prefix sys;
    reference
      "RFC 7317: A YANG Data Model for System Management";
```

```
  }

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

     Editor:   Qiufang Ma
               <mailto:maqiufang1@huawei.com
     Author:   Qin Wu
               <mailto:bill.wu@huawei.com>
     Editor:   Mohamed Boucadair
               <mailto:mohamed.boucadair@orange.com>
     Author:   Daniel King
               <mailto:d.king@lancaster.ac.uk>";
  description
    "This YANG module defines two groupings for iCalendar (Internet
     Calendaring and Scheduling Core Object Specification) data
     types: period of time and recurrence rule, for representing and
     exchanging calendaring and scheduling information. The YANG
     module complies with Sections 3.3.9 and 3.3.10 of RFC 5545.

     Copyright (c) 2024 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.

     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 2023-01-19 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: A YANG Data Model for Scheduling";
```

```
  }

  feature basic-recurrence-supported {
    description
      "Indicates that the server supports configuring a basic
       scheduled recurrence.";
  }

  feature icalendar-recurrence-supported {
    description
      "Indicates that the server supports configuring a comprehensive
       scheduled icalendar recurrence";
    reference
      "RFC 5545: Internet Calendaring and Scheduling Core Object
                 Specification (iCalendar),
                 Sections 3.3.10 and 3.8.5";
  }

  typedef weekday {
    type enumeration {
      enum sunday {
        value 0;
        description
          "Sunday of the week.";
      }
      enum monday {
        value 1;
        description
          "Monday of the week.";
      }
      enum tuesday {
        value 2;
        description
          "Tuesday of the week.";
      }
      enum wednesday {
        value 3;
        description
          "Wednesday of the week.";
      }
      enum thursday {
        value 4;
        description
          "Thursday of the week.";
      }
      enum friday {
        value 5;
        description
```

```
            "Friday of the week.";
        }
        enum saturday {
          value 6;
          description
            "Saturday of the week.";
        }
      }
      description
        "Seven days of the week.";
    }

    typedef duration {
      type string {
        pattern '((\+)?|\-)P((([0-9]+)D)?(T(0[0-9]|1[0-9]|2[0-3])'
              + ':[0-5][0-9]:[0-5][0-9]))|P([0-9]+)W';
      }
      description
        "Duration of the time. The format can represent nominal
         durations (weeks designated by 'W' and days designated by 'D')
         and accurate durations (hours:minutes:seconds follows the
         designator 'T').

         Note that this value type doesn't support the 'Y' and 'M'
         designators to specify durations in terms of years and months.

         Negative durations are typically used to schedule an alarm to
         trigger before an associated time.";
      reference
        "RFC 5545: Internet Calendaring and Scheduling Core Object
                   Specification (iCalendar), Sections 3.3.6 and
                   3.8.6.3";
    }

    identity frequency-type {
      description
        "Base identity for frequency type.";
    }
    identity secondly {
      base frequency-type;
      description
        "Identity for a repeating event based on an interval of
         a second or more.";
    }
    identity minutely {
      base frequency-type;
      description
        "Identity for a repeating event based on an interval of
```

```
        a minute or more.";
    }
    identity hourly {
      base frequency-type;
      description
        "Identity for a repeating event based on an interval of
         an hour or more.";
    }
    identity daily {
      base frequency-type;
      description
        "Identity for a repeating event based on an interval of
         a day or more.";
    }
    identity weekly {
      base frequency-type;
      description
        "Identity for a repeating event based on an interval of
         a week or more.";
    }
    identity monthly {
      base frequency-type;
      description
        "Identity for a repeating event based on an interval of
         a month or more.";
    }
    identity yearly {
      base frequency-type;
      description
        "Identity for a repeating event based on an interval of
         a year or more.";
    }
    identity schedule-type {
      description
        "Base identity for schedule type.";
    }
    identity period {
      base schedule-type;
      description
        "Identity for a period based schedule.";
    }

    identity recurrence {
      base schedule-type;
      description
        "Identity for a recurrence based schedule.";
    }
    identity schedule-state {
```

```
      description
        "Base identity for schedule state.";
    }
    identity enabled {
      description
        "Identity for the recurrence with enabled state.";
    }
    identity disabled {
      description
        "Identity for the recurrence with disabled state.";
    }
    identity out-of-date {
      description
        "Identity for the recurrence with out-of-date state.";
    }

    grouping generic-schedule-params {
      description
        "Incldues a set of generic configuration parameters that are
         followed by the entity that supports schedules.

         These parameters apply to all schedules.

          Such parameters are used as guards to prevent, e.g., stale
          configuration.";
      leaf time-zone-identifier {
        type sys:timezone-name;
        description
          "Indicates the identifier for the time zone in a time zone
           database.";
      }
      leaf validity {
        type yang:date-and-time;
        description
          "Specifies the date and time after which a schedule will
           be considered as invalid. This paratemer takes precedence
           over similar attributes that are provided at the schedule
           instance itself.";
      }
      leaf max-allowed-start {
        type yang:date-and-time;
        description
          "Specifies the date and time after which a requested schedule
           instance cannot be accepted by the entity. Specifically,
           a requested schedule will be rejected if the first occurence
           of that new schedule exceeds 'max-allowed-start'.";
      }
      leaf min-allowed-start {
```

```
        type yang:date-and-time;
        description
          "Specifies the date and time before which a requested
           schedule instance cannot be accepted by the entity.
           Specifically, a requested schedule will be rejected if the
           first occurence of that new schedule is to be scheduled
           before 'min-allowed-start'.";
      }
      leaf max-allowed-end {
        type yang:date-and-time;
        description
          "A requested schedule will be rejected if the last
           occurence of that schedule or its duratioon exceed
           'max-allowed-end'.";
      }
      leaf discard-action {
        type enumeration {
          enum warning {
            description
              "A warning message is generated when a schedule is
               discarded when enforcing the guards in this grouping
               or it is received out-of-date.";
          }
          enum silently-discard {
            description
             "Discards silently a schedule when it is invalid because
              it is not consistent with the guards in this grouping or
              it is received out-of-date.";
          }
        }
        description
          "Specifies the behavior when a schedule is discarded.";
      }
    }

    grouping period-of-time {
      description
        "This grouping is defined for period of time property.";
      reference
        "RFC 5545: Internet Calendaring and Scheduling Core Object
                   Specification (iCalendar), Section 3.3.9";
      leaf period-start {
        type yang:date-and-time;
        description
          "Period start time.";
      }
      leaf time-zone-identifier {
        type sys:timezone-name;
```

```
        description
          "Indicates the identifier for the time zone in a time zone
           database. This parameter MUST be specified if 'period-start'
           value is neither reported in the format of UTC nor time zone
           offset to UTC.";
      }
      choice period-type {
        description
          "Indicates the type of the time period. Two types are
            supported.";
        case explicit {
          description
            "A period of time is identified by its start and its end.
             'period-start' indicates the period start.";
          leaf period-end {
            type yang:date-and-time;
            description
              "Period end time. The start MUST be before the end. If a
               local time without time zone offset to UTC time is
               specified, it MUST use the same time zone reference as
               'period-start' parameter. If 'period-start' also uses a
               local time without time zone offset to UTC, it MUST use
               the time zone as specified by the
               'time-zone-identifier' parameter.";
          }
        }
        case duration {
          description
            "A period of time is defined by a start and a
             positive duration of time.";
          leaf duration {
            type duration {
              pattern 'P((([0-9]+)D)?(T(0[0-9]|1[0-9]|2[0-3])'
                    + ':[0-5][0-9]:[0-5][0-9]))|P([0-9]+)W';
            }
            description
              "A positive duration of the time. This value is
               equivalent to the format of duration type except that
               the value cannot be negative.";
          }
        }
      }
    }

    grouping recurrence {
      description
        "A simple definition of recurrence.";
      container recurrence-first {
```

```
        description
          "Specifies the first instance of the recurrence.";
        leaf date-time-start {
          type union {
            type yang:date-no-zone;
            type yang:date-and-time;
          }
          description
            "Defines the first instance in the recurrence set. If it is
             specified as a date-no-zone value type with no duration
             specified, the recurrence's duration is taken to be one
             day.";
          reference
            "RFC 5545: Internet Calendaring and Scheduling Core Object
                       Specification (iCalendar), Section 3.3.10";
        }
        leaf time-zone-identifier {
          type sys:timezone-name;
          description
            "Indicates the identifier for the time zone in a time zone
             database. This parameter MUST be specified if 'start'
             value is neither reported in the format of UTC nor time
             zone offset to UTC.";
        }
        leaf duration {
          type duration;
          description
            "When specified, it refers to the duration of
             the first occurrence. The exact duration also applies to
             all the recurrence instance.";
        }
      }
      leaf frequency {
        type identityref {
          base frequency-type;
        }
        description
          "This parameter is defined to identify the frequency type of
           the recurrence rule.";
      }
      leaf interval {
        type uint32;
        description
          "A positive integer representing at which intervals the
           recurrence rule repeats. For example, within a 'daily'
           recurrence rule, a value of '8' means every eight days.";
      }
      choice recurrence-bound {
```

```
        description
          "Modes to bound the recurrence rule. If no choice is
           indicated, the recurrence rule is considered to repeat
           forever.";
        case until {
          description
            "This case defines a way that bounds the recurrence
             rule in an inclusive manner.";
          leaf until {
            type union {
              type yang:date-no-zone;
              type yang:date-and-time;
            }
            description
              "This parameter specifies a date-no-zone or
               date-time value to bounds the recurrence. If the value
               specified by this parameter is synchronized with the
               specified recurrence, it becomes the last instance of
               the recurrence. The value MUST have the same value type
               as the value type of 'start' parameter.";
          }
        }
        case count {
          description
            "This case defines the number of occurrences at which
             to range-bound the recurrence.";
          leaf count {
            type uint32;
            description
              "The positive number of occurrences at which to
               range-bound the recurrence.";
          }
        }
      }
    }

    grouping recurrence-with-date-times {
      description
        "This grouping defines an aggregate set of repeating
         occurrences. The recurrence instances are defined by
         the union of occurrences defined by both the
         'recurrence' and 'date-times'. Duplicate instances
         are ignored.";
      uses recurrence;
      choice date-times-choice {
        description
          "Specify a list of occurrences which complement the
           recurrence set defined by 'recurrence' grouping. If
```

```
           it is specified as a period value, the duration of
           the recurrence instance will be the one specified
           by it, and not the duration defined inside the
           recurrence-first parameter.";
        case date-time {
          description
            "Specify a list of occurrences with date-and-time
             values.";
            leaf-list date-times {
              type yang:date-and-time;
              description
                "Specifies a set of date-and-time values of
                 occurrences.";
            }
        }
        case date {
          description
            "Specifies a list of occurrences with date-no-zone
             values.";
            leaf-list dates {
              type yang:date-no-zone;
              description
                "Specifies a set of date-no-zone values of
                 occurrences.";
            }
        }
        case period-timeticks {
          description
            "Specifies a list of occurrences with period span of
             timeticks format.";
            list period-timeticks {
              key "period-start";
              description
                "A list of period with timeticks formats.";
              leaf period-start {
                type yang:timeticks;
                must
                "(not(derived-from(../../frequency,"
               +"'schedule:secondly')) or (current() < 100)) and "
               +"(not(derived-from(../../frequency,"
               +"'schedule:minutely')) or (current() < 6000)) and "
               +"(not(derived-from(../../frequency,'schedule:hourly'))"
               +" or (current() < 360000)) and "
               +"(not(derived-from(../../frequency,'schedule:daily'))"
               +" or (current() < 8640000)) and "
               +"(not(derived-from(../../frequency,'schedule:weekly'))"
               +" or (current() < 60480000)) and "
               +"(not(derived-from(../../frequency,"
```

```
                +"'schedule:monthly')) or (current() < 267840000)) and "
                +"(not(derived-from(../../frequency,'schedule:yearly'))"
                +" or (current() < 3162240000))" {
                   error-message
                     "The period-start must not exceed the frequency
                      interval.";
                 }
                 description
                   "Start time of the scheduled value within one
                    recurrence.";
               }
               leaf period-end {
                 type yang:timeticks;
                 description
                   "End time of the scheduled value within one
                    recurrence.";
               }
             }
         }
         case period {
           description
             "Specifies a list of occurrences with period span
              of date-and -time format.";
           list period {
             key "period-start";
             description
               "A list of period with date-and-time formats.";
             uses period-of-time;
           }
         }
       }
     }

     grouping icalendar-recurrence {
       description
         "This grouping is defined to identify properties
          that contain a recurrence rule.";
       reference
         "RFC 5545: Internet Calendaring and Scheduling
                    Core Object Specification (iCalendar),
                    Section 3.8.5";

       uses recurrence-with-date-times;
       leaf-list bysecond {
         type uint32 {
           range "0..60";
         }
         description
```

```
          "A list of seconds within a minute.";
      }
      leaf-list byminute {
        type uint32 {
          range "0..59";
        }
        description
          "A list of minutes within an hour.";
      }
      leaf-list byhour {
        type uint32 {
          range "0..23";
        }
        description
          "Specifies a list of hours of the day.";
      }
      list byday {
        key "weekday";
        description
          "Specifies a list of days of the week.";
        leaf-list direction {
          when "derived-from(../../frequency, 'schedule:monthly') or "
            +  "(derived-from(../../frequency, 'schedule:yearly') "
            +  " and not(../../byyearweek))";
          type int32 {
            range "-53..-1|1..53";
          }
          description
            "When specified, it indicates the nth occurrence of a
             specific day within the monthly or yearly recurrence
             rule.
             For example, within a monthly rule, +1 monday represents
             the first monday within the month, whereas -1 monday
             represents the last monday of the month.";
        }
        leaf weekday {
          type schedule:weekday;
          description
            "Corredponds to seven days of the week.";
        }
      }

      leaf-list bymonthday {
        type int32 {
          range "-31..-1|1..31";
        }
        description
          "Specifies a list of days of the month.";
```

```
      }
      leaf-list byyearday {
        type int32 {
          range "-366..-1|1..366";
        }
        description
          "Specifies a list of days of the year.";
      }
      leaf-list byyearweek {
        when "derived-from(../frequency, 'schedule:yearly')";
        type int32 {
          range "-53..-1|1..53";
        }
        description
          "Specifies a list of weeks of the year.";
      }
      leaf-list byyearmonth {
        type uint32 {
          range "1..12";
        }
        description
          "Specifies a list of months of the year.";
      }
      leaf-list bysetpos {
        type int32 {
          range "-366..-1|1..366";
        }
        description
          "Specifies a list of values that corresponds to the nth
           occurrence within the set of recurrence instances
           specified by the rule. It must only be used in conjunction
           with another by the rule part.";
      }
      leaf workweek-start {
        type schedule:weekday;
        default "monday";
        description
          "Specifies the day on which the workweek starts.";
      }
      leaf-list exception-dates {
        type union {
          type yang:date-no-zone;
          type yang:date-and-time;
        }
        description
          "Defines a list of exceptions for recurrence.";
      }
    }
```

```
    grouping schedule-status {
      description
        "This grouping is defined to identify common properties of
         scheduling status.";
      leaf schedule-id {
        type string;
        description
          "The schedule identifier that globally identifies a
           schedule in a device, controller, network, etc.
           The unicity scope depends on the implementation.";
      }
      leaf state {
        type identityref {
          base schedule-state;
        }
        description
          "The current state of the schedule.";
      }
      leaf version {
        type uint16;
        description
          "The version number of the schedule.";
      }
      leaf schedule-type {
        type identityref {
          base schedule-type;
        }
        config false;
        description
          "The schedule type.";
      }
      leaf last-update {
        type yang:date-and-time;
        config false;
        description
          "The timestamp that the schedule is last updated.";
      }
      leaf counter {
        when "derived-from-or-self(../schedule-type, "
           + "'schedule:recurrence')";
        type uint32;
        config false;
        description
          "The counter of occurrences since the schedule was enabled.
           The count wraps around when it reaches the maximum value.";
      }
      leaf last-occurrence {
        when "derived-from-or-self(../schedule-type, "
```

```
            + "'schedule:recurrence')";
          type yang:date-and-time;
          config false;
          description
            "The timestamp of last occurrence.";
        }
        leaf upcoming-occurrence {
          when "derived-from-or-self(../schedule-type, "
             + "'schedule:recurrence')"
             + "and derived-from-or-self(../state, 'schedule:enabled')";
          type yang:date-and-time;
          config false;
          description
            "The timestamp of next occurrence.";
        }
      }
    }
    <CODE ENDS>
```

7. Security Considerations

This section uses the template described in Section 3.7 of [I-D.ietf-netmod-rfc8407bis].

The "ietf-schedule" YANG module 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.

The "ietf-schedule" module defines a set of types and groupings. These nodes are intended to be reused by other YANG modules. The module by itself does not expose any data nodes that are writable, data nodes that contain read-only state, or RPCs. As such, there are no additional security issues related to the "ietf- schedule" module that need to be considered.

8. IANA Considerations

8.1. The "IETF XML" Registry

This document registers the following URI in the "IETF XML Registry" [RFC3688].

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

8.2. The "YANG Module Names" Registry

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

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

9. References

9.1. Normative References

[I-D.ietf-netmod-rfc6991-bis]
Schönwälder, J., "Common YANG Data Types", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc6991-bis-15, 23 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-rfc6991-bis-15>.

[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>.

[RFC5545] Desruisseaux, B., Ed., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, DOI 10.17487/RFC5545, September 2009, <https://www.rfc-editor.org/info/rfc5545>.

[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>.

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

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

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[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>.

9.2. Informative References

[I-D.contreras-opsawg-scheduling-oam-tests]
Contreras, L. M. and V. Lopez, "A YANG Data Model for Network Diagnosis by scheduling sequences of OAM tests", Work in Progress, Internet-Draft, draft-contreras-opsawg-scheduling-oam-tests-01, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-contreras-opsawg-scheduling-oam-tests-01>.

[I-D.ietf-netmod-eca-policy]
Wu, Q., Bryskin, I., Birkholz, H., Liu, X., and B. Claise, "A YANG Data model for ECA Policy Management", Work in

Progress, Internet-Draft, draft-ietf-netmod-eca-policy-01, 19 February 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-eca-policy-01>.

[I-D.ietf-netmod-rfc8407bis]
Bierman, A., Boucadair, M., and Q. Wu, "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc8407bis-09, 28 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-rfc8407bis-09>.

[I-D.ietf-opsawg-ucl-acl]
Ma, Q., Wu, Q., Boucadair, M., and D. King, "A YANG Data Model and RADIUS Extension for Policy-based Network Access Control", Work in Progress, Internet-Draft, draft-ietf-opsawg-ucl-acl-03, 2 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-ucl-acl-03>.

[I-D.ietf-tvr-use-cases]
Birrane, E. J., Kuhn, N., Qu, Y., Taylor, R., and L. Zhang, "TVR (Time-Variant Routing) Use Cases", Work in Progress, Internet-Draft, draft-ietf-tvr-use-cases-09, 29 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-tvr-use-cases-09>.

[I-D.united-tvr-schedule-yang]
Qu, Y., Lindem, A., Kinzie, E., Fedyk, D., and M. Blanchet, "YANG Data Model for Scheduled Attributes", Work in Progress, Internet-Draft, draft-united-tvr-schedule-yang-00, 11 October 2023, <https://datatracker.ietf.org/doc/html/draft-united-tvr-schedule-yang-00>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Examples of Format Representation

This section provides some examples to illustrate the use of the period and recurrence formats defined as YANG groupings. Note that "grouping" does not define any data nodes in the schema tree, the examples illustrated are just for the ease of understanding. Only the message body is provided with JSON used for encoding [RFC7951].

A.1. The "period-of-time" Grouping

The example of a period that starts at 08:00:00 UTC, on January 1, 2025 and ends at 18:00:00 UTC on December 31, 2027 is encoded as shown in Figure 8.

```
{
  "period-start": "2025-01-01T08:00:00Z",
  "period-end": "2027-12-01T18:00:00Z"
}
```

Figure 8: Simple Start/End Schedule

An example of a period that starts at 08:00:00 UTC, on January 1, 2025 and lasts 15 days and 5 hours and 20 minutes is encoded as shown in Figure 9.

```
{
  "period-start": "2025-01-01T08:00:00Z",
  "duration": "P15DT05:20:00"
}
```

Figure 9: Simple Schedule with Duration

An example of a period that starts at 2:00 A.M. in Los Angeles on November 19, 2025 and lasts 20 weeks is depicted in Figure 10.

```
{
  "period-start": "2025-11-19T02:00:00",
  "time-zone-identifier": "America/Los_Angeles",
  "duration": "P20W"
}
```

Figure 10: Simple Schedule with Time Zone Indication

A.2. The "recurrence" Grouping

Figure 11 indicates a recurrence of every 2 days for 10 occurrences, starting at 3 p.m. on December 1, 2025 in the Eastern United States time zone:

```
{
  "recurrence-first": {
    "date-time-start": "2025-11-01T15:00:00",
    "time-zone-identifier": "America/New_York"
  },
  "frequency": "ietf-schedule:daily",
  "interval": 2,
  "count": 10
}
```

Figure 11: Simple Schedule with Recurrence

Figure 12 illustrates an example of an anniversary that will occur annually, from 1997-11-25, until 2050-11-25:

```
{
  "recurrence-first": {
    "date-time-start": "1979-11-25"
  },
  "frequency": "ietf-schedule:yearly",
  "until": "2050-11-25"
}
```

Figure 12: Simple Schedule with Recurrence and End Date

A.3. The "recurrence-with-date-times" Grouping

Figure 13 indicates a recurrence that occurs every 2 hours from 9:00 AM to 5:00 PM, and 6PM UTC time on 2025-12-01:

```
{
  "recurrence-first": {
    "date-time-start": "2025-12-01T09:00:00Z"
  },
  "frequency": "ietf-schedule:hourly",
  "interval": 2,
  "until": "2025-12-01T17:00:00Z",
  "date-times": ["2025-12-01T18:00:00Z"]
}
```

Figure 13: Example of Recurrence With Date Times

Figure 14 indicates a recurrence that occurs every 30 minutes and last for 15 minutes from 9:00 AM to 5:00 PM, and extra two occurrences at 6:00 PM and 6:30 PM with each lasting for 20 minutes on 2025-12-01:

```
{
  "recurrence-first": {
    "date-time-start": "2025-12-01T09:00:00Z",
    "duration": "PT00:15:00"
  },
  "frequency": "ietf-schedule:minutely",
  "interval": 30,
  "until": "2025-12-01T17:00:00Z",
  "period": [
    {
      "period-start": "2025-12-01T18:00:00Z",
      "duration": "PT00:20:00"
    },
    {
      "period-start": "2025-12-01T18:30:00Z",
      "duration": "PT00:20:00"
    }
   ]
}
```

Figure 14: Example of Advanced Recurrence Schedule

A.4. The "icalendar-recurrence" Grouping

Figure 15 indicates 10 occurrences that occur at 8:00 AM (EST), every last Saturday of the month starting in January 2024:

```
{
  "recurrence-first": {
    "date-time-start": "2024-01-27T08:00:00",
    "time-zone-identifier": "America/New_York"
  },
  "frequency": "ietf-schedule:monthly",
  "count": 10,
  "byday": [
    {
      "direction": [-1],
      "weekday": "saturday"
    }
  ]
}
```

Figure 15: Simple iCalendar Recurrence

Figure 16 is an example of a recurrence that occurs on the last workday of the month until December 25, 2024, from January 1, 2024:

```
{
  "recurrence-first": {
  "date-time-start": "2024-01-01"
  },
  "frequency": "ietf-schedule:monthly",
  "until": "2024-12-25",
  "byday": [
    { "weekday": "monday"},
    { "weekday": "tuesday"},
    { "weekday": "wednesday"},
    { "weekday": "thursday"},
    { "weekday": "friday"}
  ],
  "bysetpos": [-1]
}
```

Figure 16: Example of Advanced iCalendar Recurrence

Figure 17 indicates a recurrence that occur every 20 minutes from 9:00 AM to 4:40 PM (UTC), with the occurrence starting at 10:20 AM being excluded on 2025-12-01:

```
{
  "recurrence-first": {
    "date-time-start": "2025-12-01T09:00:00Z"
  },
  "until": "2025-12-01T16:40:00Z",
  "frequency": "ietf-schedule:minutely",
  "byminute": [0, 20, 40],
  "byhour": [9, 10, 11, 12, 13, 14, 15, 16],
  "exception-dates": ["2025-12-01T10:20:00Z"]
}
```

Figure 17: Example of Advanced iCalendar Recurrence with Exceptions

Appendix B. Examples of Using/Extending the "ietf-schedule" Module

This non-normative section shows two examples for how the "ietf-schedule" module can be used or extended for scheduled events or attributes based on date and time.

B.1. Example: Schedule Tasks to Execute Based on a Recurrence Rule

Scheduled tasks can be used to execute specific actions based on certain recurrence rules (e.g., every Friday at 8:00 AM). The following example module which "uses" the "icalendar-recurrence" grouping from "ietf-schedule" module shows how a scheduled task could be defined with different features used for options.

```
module example-scheduled-backup {
  yang-version 1.1;
  namespace "http://example.com/example-scheduled-backup";
  prefix "ex-scback";

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-schedule {
    prefix "schedule";
  }

  organization
    "Example, Inc.";

  contact
    "Support at example.com";

  description
    "Example of a module defining an scheduled based backup
     operation.";

  revision "2023-01-19" {
    description
      "Initial Version.";
    reference
      "RFC XXXX: A YANG Data Model for Scheduling.";
    }

  container scheduled-backup-tasks {
    description
      "A container for backing up all current running configuration
       on the device.";
    list tasks {
      key "task-id";
      description
        "The list of backing up tasks on this device.";
      leaf task-id {
        type string;
        description
          "The task identifier that uniquely identifies a scheduled
           backup task.";
      }
      choice local-or-remote {
        description
          "Specifies whether the configuration to be backed up is
           local or remote.";
```

```
          case local {
            description
              "Configuration parameters for backing up of local
               devices.";
            leaf local {
              type empty;
              description
                "The parameter specifies the configuration to be
                 backed up is on the local device.";
            }
          }
          case remote {
            description
              "Configuration parameters for backing up of remote
               devices.";
            leaf remote {
              type inet:domain-name;
              description
                "The parameter specifies the remote device domain
                 name.";
            }
          }
        }

        container basic-recurrence-schedules {
          if-feature schedule:basic-recurrence-supported;
          description
            "Basic recurrence schedule specification, only applies when
             schedule:basic-recurrence-supported feaure is supported.";
          leaf schedule-id {
            type string;
            description
              "The schedule identifier for this recurrence rule.";
          }
          uses schedule:recurrence;
         }

        container icalendar-recurrence-schedules {
          if-feature schedule:icalendar-recurrence-supported;
          description
            "Basic recurrence schedule specification, only applies when
             schedule:icalendar-recurrence-supported feaure is
             supported.";
          leaf schedule-id {
            type string;
            description
              "The schedule identifier for this recurrence rule.";
          }
```

```
          uses schedule:icalendar-recurrence;
        }
      }

      list schedule-set {
        key "schedule-id";
        description
          "The list of schedule status for the backup tasks.";
        uses schedule:schedule-status;
      }
    }
  }
```

B.2. Example: Schedule Network Properties to Change Based on Date and Time

Network properties may change over a specific period of time or based on a recurrence rule, e.g., [I-D.ietf-tvr-use-cases]. The following example module which augments the "recurrence-with-date-times" grouping from "ietf-schedule" module shows how a scheduled based attribute could be defined.

```
module example-scheduled-link-bandwidth {
  yang-version 1.1;
  namespace "http://example.com/example-scheduled-link-bandwidth";
  prefix "ex-scattr";

  import ietf-network {
    prefix "nw";
    reference
      "RFC 8345: A YANG Data Model for Network Topologies";
  }

  import ietf-schedule {
    prefix "schedule";
    reference
      "RFC XXXX: A YANG Data Model for Scheduling";
  }

  organization
    "Example, Inc.";

  contact
    "Support at example.com";

  description
    "Example of a module defining a scheduled link bandwidth.";
```

```
    revision "2023-01-19" {
      description
        "Initial Version.";
      reference
        "RFC XXXX: A YANG Data Model for Scheduling.";
      }

    grouping link-bandwidth-grouping {
      description
        "Grouping of the link bandwidth definition.";
      leaf scheduled-bandwidth {
        type uint64;
        units "Kbps";
        description
          "Bandwidth values, expressed in kilobits per second.";
      }
    }

    container link-attributes {
      description
        "Definition of link attributes.";
      list link {
        key "source-node destination-node";
        description
          "Definition of link attributes.";
        leaf source-node {
          type nw:node-id;
          description
            "Indicates the source node identifier.";
        }
        leaf destination-node {
          type nw:node-id;
          description
            "Indicates the source node identifier.";
        }

        leaf default-bandwidth {
          type uint64;
          units "Kbps";
          description
            "Default bandwidth values when unspecified.";
        }

        choice time-variant-type {
          description
            "Controls the schedule type.";
          case period {
            uses schedule:period-of-time;
```

```
          }
          case recurrence {
            uses schedule:recurrence-with-date-times {
              augment "date-times-choice/period-timeticks"
                    + "/period-timeticks" {
                description
                  "Specifies the attributes inside each
                   period-timeticks entry.";
                uses link-bandwidth-grouping;
              }
              augment "date-times-choice/period/period" {
                description
                  "Specifies the attributes within each period entry.";
                uses link-bandwidth-grouping;
              }
            }
          }
        }
      }
    }
  }
```

Figure 18 shows a configuration example of a link's bandwidth that is scheduled between 2023/12/01 0:00 UTC to the end of 2023/12/31 with a daily schedule. In each day, the bandwidth value is scheduled to be 500 Kbps between 1:00 AM to 6:00 AM and 800 Kbps between 10:00 PM to 11:00 PM. The bandwidth value that's not covered by the period above is 1000 Kbps by default.

```
<?xml version="1.0" encoding="utf-8"?>
<link-attributes
  xmlns="http://example.com/example-scheduled-link-bandwidth"
  xmlns:schedule="urn:ietf:params:xml:ns:yang:ietf-schedule">
  <link>
    <source-node>ne1</source-node>
    <destination-node>ne2</destination-node>
    <default-bandwidth>1000</default-bandwidth>
    <recurrence-first>
      <date-time-start>2023-12-01T01:00:00Z</date-time-start>
    </recurrence-first>
    <frequency>schedule:daily</frequency>
    <until>2023-12-31T23:59:59Z</until>
    <period-timeticks>
      <period-start>360000</period-start>
      <period-end>2160000</period-end>
      <scheduled-bandwidth>500</scheduled-bandwidth>
    </period-timeticks>
    <period-timeticks>
      <period-start>7920000</period-start>
      <period-end>8280000</period-end>
      <scheduled-bandwidth>800</scheduled-bandwidth>
    </period-timeticks>
  </link>
</link-attributes>
```

Figure 18: Example of Scheduled Link's Bandwidth

Acknowledgments

This work is derived from the [I-D.ietf-opsawg-ucl-acl]. There is a desire from the OPSAWG to see this model be separately defined for wide use in scheduling context.

Thanks to Adrian Farrel, Wei Pan, Tianran Zhou, and Joe Clarke for their valuable comments and inputs to this work.

Many thanks to the authors of [I-D.united-tvr-schedule-yang], [I-D.contreras-opsawg-scheduling-oam-tests], and [I-D.ietf-netmod-eca-policy] for the constructive discussion during IETF#118.

Authors' Addresses

Qiufang Ma (editor)
Huawei
101 Software Avenue, Yuhua District
Jiangsu
210012
China
Email: maqiufang1@huawei.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Jiangsu
210012
China
Email: bill.wu@huawei.com

Mohamed Boucadair (editor)
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.com

Daniel King
Lancaster University
United Kingdom
Email: d.king@lancaster.ac.uk