

Network Configuration
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

R. Wilton, Ed.
Cisco Systems
H. Keller
Deuetsche Telekom
B. Claise
Huawei
E. Aries
Juniper
J. Cumming
Nokia
T. Graf
Swisscom
20 October 2025

YANG Datastore Telemetry (YANG Push Lite)
draft-wilton-netconf-yang-push-lite-02

Abstract

YANG Push Lite is a YANG datastore telemetry solution, as an alternative specification to the Subscribed Notifications and YANG Push solution, specifically optimized for the efficient observability of operational data.

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://rgwilton.github.io/draft-yp-observability/draft-wilton-netconf-yp-observability.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-wilton-netconf-yang-push-lite/>.

Discussion of this document takes place on the Network Configuration Working Group mailing list (<mailto:netconf@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/netconf/>. Subscribe at <https://www.ietf.org/mailman/listinfo/netconf/>.

Source for this draft and an issue tracker can be found at <https://github.com/rgwilton/draft-yp-observability>.

Status of This Memo

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

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

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

This Internet-Draft will expire on 23 April 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Document Status	5
2. Conventions	6
3. Introduction	6
3.1. Background and Motivation for YANG Push Lite	6
3.2. Complexities in Modelling the Operational State Datastore	8
3.3. Complexities for Consumers of YANG Push Data	8
3.3.1. Combined periodic and on-change subscription	9
3.4. Relationships to existing RFCs and Internet Drafts	10
3.4.1. RFC 8639 and RFC 8641	10
3.4.2. I-D.draft-ietf-netconf-notif-envelope and RFC 5277	10
3.4.3. RFC 9196 and I-D.draft-netana-netconf-yp-transport-capabilities	10
3.4.4. I-D.draft-ietf-netconf-https-notif and I-D.draft-ietf-netconf-udp-notif	11
3.4.5.	
4. YANG Push Lite Overview	11
5. Definitions	13
6. Subscription paths and selection filters	16
6.1. _YPath_ definition	17

6.2.	The "filters" Container	18
6.3.	Decomposing Subscription Filters	19
7.	Datastore Event Streams	20
7.1.	Notification Envelope	20
7.2.	Event Records	22
7.3.	Types of subscription event monitoring	23
7.4.	Periodic events	24
7.5.	On-Change events	24
7.5.1.	On-Change Notifiable Datastore Nodes	25
7.5.2.	On-Change Considerations	26
7.6.	Combined periodic and on-change subscriptions	27
7.7.	Streaming Update Examples	27
8.	Receivers, Transports, and Encodings	28
8.1.	Receivers	28
8.1.1.	Receivers for Configured Subscriptions	29
8.1.2.	Receivers for Dynamic Subscriptions	31
8.1.3.	Receiver Session States and State Machine	31
8.2.	Transports	32
8.2.1.	Requirements for Yang Push Lite Transport Specifications	33
8.3.	Encodings	34
9.	Setting up and Managing Subscriptions	35
9.1.	Common Subscription Parameters	35
9.1.1.	Subscription States	36
9.1.2.	Creating Subscriptions	38
9.1.3.	Modifying Subscriptions	39
9.1.4.	Terminating Subscriptions	41
9.2.	Subscription Lifecycle Notifications	42
9.2.1.	"subscription-started"	42
9.2.2.	"subscription-terminated"	43
9.2.3.	"receiver-disconnected"	44
9.2.4.	"update-completed"	45
9.3.	Configured Subscriptions	45
9.3.1.	Creating a Configured Subscription	47
9.3.2.	Modifying a Configured Subscription	47
9.3.3.	Deleting a Configured Subscription	47
9.3.4.	Resetting a Configured Subscription	47
9.4.	Dynamic Subscriptions	48
9.4.1.	Establishing a Dynamic Subscription	49
9.4.2.	Deleting a Dynamic Subscription	50
9.4.3.	Killing a Dynamic Subscription	51
9.4.4.	RPC Failures	51
10.	Robustness, Reliability, and Subscription Monitoring	53
10.1.	Robustness and Reliability	53
10.2.	Subscription Monitoring	54
10.3.	Publisher Capacity	55
11.	Conformance and Capabilities	55
11.1.	Capabilities	55

11.2. Subscription Content Schema Identification	56
12. Core YANG Push Lite YANG Data Model	57
12.1. ietf-yp-lite YANG tree	57
12.2. ietf-yp-lite YANG Model	59
13. Configured Subscription YANG Data Model	79
13.1. ietf-yp-lite-config YANG tree	81
13.2. ietf-yp-lite-config YANG Model	83
14. Capabilities YANG Data Model	94
14.1. ietf-yp-lite-capabilities YANG tree	94
14.2. ietf-yp-lite-capabilities YANG Model	94
15. Security Considerations	100
15.1. Use of YANG Push Lite with NACM	101
15.2. Receiver Authorization	101
15.3. YANG Module Security Considerations	103
16. IANA Considerations	104
16.1. Namespace URI registrations	104
17. YANG Module Name registrations	104
Acknowledgments	105
Contributors	105
References	105
Normative References	105
Informative References	107
Appendix A. Functional changes between YANG Push Lite and YANG Push	111
A.1. Removed Functionality	111
A.2. Changed Functionality	112
A.3. Added Functionality	114
Appendix B. Subscription Errors (from RFC 8641)	114
B.1. RPC Failures	114
B.2. Failure Notifications	116
Appendix C. Examples	116
C.1. Example of periodic update messages	117
C.2. Example of an on-change-update notification using the new style update message	119
C.3. Example of an on-change-delete notification using the new style update message	121
C.3.1. Update message with single deleted data node	121
C.3.2. Update message with multiple on-change deletes	122
C.4. Subscription RPC examples (from RFC 8641)	123
C.4.1. "establish-subscription" RPC	123
C.4.2. "delete-subscription" RPC	126
C.4.3. "resync-subscription" RPC	126
Appendix D. Summary of Open Issues & Potential Enhancements	126
D.1. Issues related to general IETF process	126
D.2. Issue related to Terminology/Definitions	127
D.3. Issues related to YANG Push Lite Overview	127
D.4. Issues related to Subscription Paths and Selection Filters	127

D.5. Issues related to Datastore Event Streams	128
D.5.1. Further refinements on how a subscription can be decomposed internally into child subscriptions with the data returned for each child subscription:	128
D.5.2. Questions/comments on the notification message format:	129
D.6. Issues related to Receivers, Transports, & Encodings	129
D.6.1. Constraints on supporting multiple receivers:	129
D.6.2. Issues related to Transports:	130
D.7. Issues related to Setting up & Managing Subscriptions	131
D.7.1. Issues related to the configuration model:	131
D.7.2. Issues related to dynamic subscriptions:	132
D.8. Issues related to Subscription Lifecycle	133
D.9. Issues related to Performance, Reliability & Subscription Monitoring	135
D.9.1. Issues/questions related to operational data:	135
D.10. Issues related to Conformance and Capabilities	136
D.11. Issues related to the YANG Modules	136
D.12. Issues related to the Security Considerations (& NACM filtering)	137
D.13. Issues related to the IANA	137
D.14. Issues related to the Appendixes	137
D.14.1. Examples related issues/questions:	137
D.15. Summary of closed/resolved issues	137
D.16. Changes	138
Authors' Addresses	138

1. Document Status

RFC Editor: If present, please remove this section before publication.
RFC Editor: Please replace 'RFC XXXX' with the RFC number for this RFC.

Based on the feedback received during the IETF 121 NETCONF session, this document has currently been written as a self-contained lightweight protocol and document replacement for [RFC8639] and [RFC8641], defining a separate configuration data model.

The comparison between YANG Push and YANG Push Lite is now in Appendix A.

Open issues are either now being tracked inline in the text or in Appendix D for the higher level issues.

2. 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.

All `_YANG` tree diagrams_ used in this document follow the notation defined in [RFC8340].

3. Introduction

[I-D.ietf-nmop-yang-message-broker-integration] describes an architecture for how YANG datastore telemetry, e.g., [RFC8641], can be integrated effectively with message brokers, e.g., [Kafka], that forms part of a wider architecture for a `_Network Anomaly Detection Framework_`, specified in [I-D.ietf-nmop-network-anomaly-architecture].

This document specifies "YANG Push Lite", an lightweight alternative to Subscribed Notifications [RFC8639] and YANG Push [RFC8641]. YANG Push Lite is a separate YANG datastore telemetry solution, which can be implemented independently or, if desired, alongside [RFC8639] and [RFC8641].

At a high level, YANG Push Lite is designed to solve a similar set of requirements as YANG Push, and it reuses a significant subset of the ideas and base solution from YANG Push. YANG Push Lite defines a separate data model to allow concurrent implementation of both protocols and to facilitate more significant changes in behavior, but many of the data nodes are taken from YANG Push and have the same, or very similar definitions.

The following sections give the background for the solution, and highlight the key ways that this specification differs from the specifications that it is derived from.

3.1. Background and Motivation for YANG Push Lite

A push based telemetry solution, as described both in this document and also the YANG Push solution described by [RFC8639] and [RFC8641], is beneficial because it allows operational data to be exported by publishers more immediately and efficiently compared to legacy poll based mechanisms, such as SNMP [RFC3411]. Some further background information on the general motivations for push based telemetry, which equally apply here, can be found in the `_Motivation_` (section 1.1) of [RFC8639] and the Introduction (section 1) of [RFC8641]. The

remainder of this section is focused on the reasons why a new lightweight version of YANG Push has been specified, and what problems it aims to solve.

Early implementation efforts of the [I-D.ietf-nmop-yang-message-broker-integration] architecture hit issues with using either of the two common YANG datastore telemetry solutions that have been specified, i.e., [gNMI] or YANG Push [RFC8641].

gNMI is specified by the OpenConfig Industry Consortium. It is more widely implemented, but operators report that some inter-operability issues between device implementations cause problems. Many of the OpenConfig protocols and data models are also expected to evolve more rapidly than IETF protocols and models - that are expected to have a more gradual pace of evolution once an RFC has been published.

YANG Push [RFC8641] was standardized by the IETF in 2019, but market adoption has been rather slow. During 2023/2024, when vendors started implementing, or considering implementing, YANG Push, it was seen that some design choices for how particular features have been specified in the solution make it expensive and difficult to write performant implementations, particularly when considering the complexities and distributed nature of operational data. In addition, some design choices of how the data is encoded (e.g., YANG Patch [RFC8072]) make more sense when considering changes in configuration data but less sense when the goal is to export a subset of the operational data off the device in an efficient fashion for both devices (publishers) and clients (receivers).

Hence, during 2024, the vendors and operators working towards YANG telemetry solutions agreed to a plan to implement a subset of [RFC8639] and [RFC8641], including common agreements of features that are not needed and would not be implemented, and deviations from the standards for some aspects of encoding YANG data. In addition, the implementation efforts identified the minimal subset of functionality needed to support the initial telemetry use cases, and areas of potential improvement and optimization to the overall YANG Push telemetry solution (which has been written up as a set of small internet drafts that augment or extend the base YANG Push solution).

Out of this work, consensus was building to specify a cut down version of Subscribed Notifications [RFC8639] and YANG Push [RFC8641] that is both more focussed on the operational telemetry use case and is also easier to implement, achieved by relaxing some of the constraints on consistency on the device, and removing, or simplifying some of the operational features. This has resulted in this specification, YANG Push Lite.

The implementation efforts also gave rise to potential improvements to the protocol and encoding of notification messages.

3.2. Complexities in Modelling the Operational State Datastore

The YANG abstraction of a single datastore of related consistent data works very well for configuration that has a strong requirement to be self consistent, and that is always updated, and validated, in a transactional way. But for producers of telemetry data, the YANG abstraction of a single operational datastore is not really possible for devices managing a non-trivial quantity of operational data.

Some systems may store their operational data in a single logical database, yet it is less likely that the operational data can always be updated in a transactional way, and often for memory efficiency reasons such a database does not store individual leaves, but instead semi-consistent records of data at a container or list entry level.

For other systems, the operational information may be distributed across multiple internal nodes (e.g., linecards), and potentially many different process daemons within those distributed nodes. Such systems generally do not, and cannot, exhibit full consistency [Consistency] of the operational data (which would require transactional semantics across all daemons and internal nodes), only offering an eventually consistent [EventualConsistency] view of the data instead.

In practice, many network devices will manage their operational data as a combination of some data being stored in a central operational datastore, and other, higher scale, and potentially more frequently changing data (e.g., statistics or FIB information) being stored elsewhere in a more memory efficient and performant way.

3.3. Complexities for Consumers of YANG Push Data

For the consumer of the telemetry data, there is a requirement to associate a schema with the instance-data that will be provided by a subscription. One approach is to fetch and build the entire schema for the device, e.g., by fetching YANG library, and then use the subscription XPath to select the relevant subtree of the schema that applies only to the subscription. The problem with this approach is that if the schema ever changes, e.g., after a software update, then it is reasonably likely of some changes occurring with the global device schema even if there are no changes to the schema subtree under the subscription path. Hence, it would be helpful to identify and version the schema associated with a particular subscription path, and also to encoded the instance data relatively to the subscription path rather than as an absolute path from the root of

the operational datastore.

TODO More needs to be added here, e.g., encoding, on-change considerations. Splitting subscriptions up.

This document proposes a new opt-in YANG-Push encoding format to use instead of the "push-update" and "push-change-update" notifications defined in [RFC8641].

1. To allow the device to split a subscription into smaller child subscriptions for more efficient independent and concurrent processing. I.e., reusing the ideas from [I-D.ietf-netconf-distributed-notif]. However, all child subscriptions are still encoded from the same subscription point.

3.3.1. Combined periodic and on-change subscription

Sometimes it is helpful to have a single subscription that covers both periodic and on-change notifications.

There are two ways in which this may be useful:

1. For generally slow changing data (e.g., a device's physical inventory), then on-change notifications may be most appropriate. However, in case there is any lost notification that isn't always detected, for any reason, then it may also be helpful to have a slow cadence periodic backup notification of the data (e.g., once every 24 hours), to ensure that the management systems should always eventually converge on the current state in the network.
2. For data that is generally polled on a periodic basis (e.g., once every 10 minutes) and put into a time series database, then it may be helpful for some data trees to also get more immediate notifications that the data has changed. Hence, a combined periodic and on-change subscription, would facilitate more frequent notifications of changes of the state, to reduce the need of having to always wait for the next periodic event.

Hence, this document introduces the fairly intuitive "periodic-and-on-change" update trigger that creates a combined periodic and on-change subscription, and allows the same parameters to be configured. For some use cases, e.g., where a time-series database is being updated, the new encoding format proposed previously may be most useful.

3.4. Relationships to existing RFCs and Internet Drafts

This document, specifying YANG Push Lite, is intended to be a lightweight alternative for [RFC8639] and [RFC8641], but that also incorporates various extensions since those RFCs were written. Often substantial parts of those documents and models have been incorporated almost verbatim, but modified to fit the YANG Push Lite functionality and module structure.

Hence, the authors of this draft would like to sincerely thank and acknowledge the very significant effort put into those RFCs and drafts by authors, contributors and reviewers. In particular, We would like to thank the listed authors of these documents: Eric Voit, Alex Clemm, Alberto Gonzalez Prieto, Einar Nilsen-Nygaard, Ambika Prasad Tripathy, Balazs Lengyel, Alexander Clemm, Benoit Claise, Qin Wu, Qiufang Ma, Alex Huang Feng, Thomas Graf, Pierre Francois.

3.4.1. RFC 8639 and RFC 8641

This document is primarily intended to be a lightweight alternative for [RFC8639] and [RFC8641], but it intentionally reuses substantial parts of the design and data model of those RFCs.

YANG Push Lite is defined using a separate module namespace, and hence can be implemented independently or, if desired, alongside [RFC8639] and [RFC8641], and the various extensions to YANG Push.

A more complete description of the main differences in YANG Push Lite compares to [RFC8639] and [RFC8641] is given in Appendix A.

3.4.2. [I-D.draft-ietf-netconf-notif-envelope] and RFC 5277

All of the notifications defined in this specification, i.e., both the datastore update message and subscription lifecycle update notifications (Section 9.2) depend upon and use the notification envelope format defined in [I-D.draft-ietf-netconf-notif-envelope].

As such, this specification does not make any use of the notification format defined in [RFC5277], but this does not prevent implementations using [RFC5277] format notifications for other YANG notifications, e.g., for the "NETCONF" event stream defined in [RFC5277].

3.4.3. RFC 9196 and [I-D.draft-netana-netconf-yp-transport-capabilities]

This document uses the capabilities concepts defined in [RFC9196].

In particular, it augments into the `ietf-system-capabilities` YANG module, but defines an equivalent alternative capability structure for the `ietf-notification-capabilities` YANG module, which defines the capabilities for YANG Push [RFC8641].

The generic transport capabilities defined in [I-D.draft-netana-netconf-yp-transport-capabilities] have been incorporated into the `ietf-yp-lite` YANG module, to augment YANG Push Lite transport capabilities and to use the different identities.

3.4.4. [I-D.draft-ietf-netconf-https-notif] and [I-D.draft-ietf-netconf-udp-notif]

The `ietf-yp-lite` YANG module has subsumed and extended the `_receivers_` data model defined in the `ietf-subscribed-notif-receivers` YANG module defined in [I-D.draft-ietf-netconf-https-notif].

The overall YANG Push Lite solution anticipates and requires new versions of both of these transports documents that augment into the `_receivers/receiver/transport-type_` choice statement, and also augment the transport identity defined in the `ietf-yp-lite` data model.

3.4.5. [I-D.draft-ietf-netconf-distributed-notif]

TODO. It is likely that some of the base support for distributed notifications will be incorporated into this draft. If so, add acknowledgements to the authors.

4. YANG Push Lite Overview

This document specifies a lightweight telemetry solution that provides a subscription service for updates to the state and changes in state from a chosen datastore.

Subscriptions specify when notification messages (also referred to as `_updates_`) should be sent, what data to include in the update records, and where those notifications should be sent.

A YANG Push lite subscription comprises:

- * a target datastore for the subscription, where the monitored subscription data is logically sourced from.
- * a set of selection filters to choose which datastore nodes from the target datastore the subscription is monitoring or sampling, as described in Section 6.

- * a choice of how update event notifications for the datastore's data nodes are triggered. I.e., either periodic sampling of the current state, on-change event-driven, or both. These are described in *TODO, add reference*.
- * a choice of encoding of the messages, e.g., JSON, or CBOR.
- * a set of one or more receivers for which datastore updates and subscription notifications are sent, as described in Section 8.1;
 - for configured subscriptions, the receivers parameters are configured, and specify transport, receiver, and encoding parameters.
 - for dynamic subscriptions, the receiver uses the same transport session on which the dynamic subscription has been created.

If a subscription is valid and acceptable to the publisher, and if a suitable connection can be made to one or more receivers associated with a subscription, then the publisher will enact the subscription, periodically sampling or monitoring changes to the chosen datastore's data nodes that match the selection filter. Push updates are subsequently sent by the publisher to the receivers, as per the terms of the subscription.

Subscriptions may be set up in two ways: either through configuration - or YANG RPCs to create and manage dynamic subscriptions. These two mechanisms are described in Section 9.

Changes to the state of subscription are notified to receivers as subscription lifecycle notifications. These are described in Section 9.2.

Security access control mechanisms, e.g., NACM {RFC8341}} can be used to ensure the receivers only get access to the information for which they are allowed. This is further described in Section 15.

While the functionality defined in this document is transport agnostic, transports like the Network Configuration Protocol (NETCONF) [RFC6241] or RESTCONF [RFC8040] can be used to configure or dynamically signal subscriptions. In the case of configured subscription, the transport used for carrying the subscription notifications is entirely independent from the protocol used to configure the subscription, and other transports, e.g., [I-D.draft-ietf-netconf-udp-notif] defines a simple UDP based transport for Push notifications. Transport considerations are described in Section 8.2. *TODO the reference to draft-ietf-netconf-udp-notif isn't right, it wouldn't be that draft, but a -bis version of it. James is querying whether we need this at all*

TODO Introduce capabilities and operational monitoring

This document defines a YANG data model, that includes RPCs and notifications, for configuring and managing subscriptions and associated configuration, and to define the format of a `_update_` notification message. The YANG model is defined in Section 12.2 and associated tree view in Section 12.1. The YANG data model defined in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].

5. Definitions

This document reuses the terminology defined in [RFC7950], [RFC8341], [RFC8342], [RFC8639] and [RFC8641].

The following terms are taken from [RFC8342]:

- * `_Datastore_`: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof. A datastore maps to an instantiated YANG data tree.
- * `_Client_`: An entity that can access YANG-defined data on a server, over some network management protocol.
- * `_Configuration_`: Data that is required to get a device from its initial default state into a desired operational state. This data is modeled in YANG using "config true" nodes. Configuration can originate from different sources.
- * `_Configuration datastore_`: A datastore holding configuration.

The following terms are taken from [RFC8639]:

- * _Configured subscription_: A subscription installed via configuration into a configuration datastore.
- * _Dynamic subscription_: A subscription created dynamically by a subscriber via a Remote Procedure Call (RPC).
- * _Event_: An occurrence of something that may be of interest. Examples include a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.
- * _Event occurrence time_: A timestamp matching the time an originating process identified as when an event happened.
- * _Event record_: A set of information detailing an event.
- * _Event stream_: A continuous, chronologically ordered set of events aggregated under some context.
- * _Event stream filter_: Evaluation criteria that may be applied against event records in an event stream. Event records pass the filter when specified criteria are met.
- * _Notification message_: Information intended for a receiver indicating that one or more events have occurred.
- * _Publisher_: An entity responsible for streaming notification messages per the terms of a subscription.
- * _Receiver_: A target to which a publisher pushes subscribed event records. For dynamic subscriptions, the receiver and subscriber are the same entity.
- * _Subscriber_: A client able to request and negotiate a contract for the generation and push of event records from a publisher. For dynamic subscriptions, the receiver and subscriber are the same entity.

The following terms are taken from [RFC8641]:

- * _Datastore node_: A node in the instantiated YANG data tree associated with a datastore. In this document, datastore nodes are often also simply referred to as "objects".
- * _Datastore node update_: A data item containing the current value of a datastore node at the time the datastore node update was created, as well as the path to the datastore node.

- * _Datastore subscription_: A subscription to a stream of datastore node updates.
- * _Datastore subtree_: A datastore node and all its descendant datastore nodes.
- * _On-change subscription_: A datastore subscription with updates that are triggered when changes in subscribed datastore nodes are detected.
- * _Periodic subscription_: A datastore subscription with updates that are triggered periodically according to some time interval.
- * _Selection filter_: Evaluation and/or selection criteria that may be applied against a targeted set of objects.
- * _Update record_: A representation of one or more datastore node updates. In addition, an update record may contain which type of update led to the datastore node update (e.g., whether the datastore node was added, changed, or deleted). Also included in the update record may be other metadata, such as a subscription ID of the subscription for which the update record was generated. In this document, update records are often also simply referred to as "updates".
- * _Update trigger_: A mechanism that determines when an update record needs to be generated.
- * _YANG-Push_: The subscription and push mechanism for datastore updates that is specified in [RFC8641].

This document introduces the following terms:

- * _Subscription_: A registration with a publisher, stipulating the information that one or more receivers wish to have pushed from the publisher without the need for further solicitation.
- * _Subscription Identifier_: A numerical identifier for a configured or dynamic subscription. Also referred to as the subscription-id.
- * _YANG-Push-Lite_: The light weight subscription and push mechanism for datastore updates that is specified in this document. *Add comment*

6. Subscription paths and selection filters

A key part of a subscription is to select which data nodes should be monitored, and so a subscription must specify both the selection filters and the datastore against which these selection filters will be applied. This information is used to choose, and subsequently push, `_update_` notifications from the publisher's datastore(s) to the subscription's receiver(s).

Filters can either be defined inline within a configured subscription (Figure 20), a dynamic subscription's `_establish-subscription_` RPC (Figure 14), or as part of the `_datastore-telemetry/filters_` container (Figure 1) which can then be referenced from a configured or dynamic subscription.

The following selection filter types are included in the YANG Push Lite data model and may be applied against a datastore:

- * `_YPaths_`: A list of basic YANG path selection filters that defines a path to a subtree of data nodes in the data tree, with some simple constraints on keys. See Section 6.1.
- * `_subtree_`: A subtree selection filter identifies one or more datastore subtrees. When specified, `_update_` records will only include the datastore nodes of selected datastore subtree(s). The syntax and semantics correspond to those specified in [RFC6241], Section 6.
- * `_XPath_`: A list of `_XPath_` ([XPath]) selection filter expressions. When specified, updates will only come from the selected datastore nodes that match the node set associated with the XPath expression.

These filters are used as selectors that define which data nodes fall within the scope of a subscription. A publisher MUST support YPath filters, and MAY also support subtree or XPath filters.

For both YPath and XPath based filters, each filter may define a list of path expressions. Each of these filter path expressions MAY be processed by the publisher independently, and if two or more filter path expressions end up selecting overlapping data nodes then the publisher MAY notify duplicate values for those data nodes, but the encoded data that is returned MUST always be syntactically valid, i.e., as per section 5.3 of [RFC8342].

6.1. `_YPath` definition

A `_YPath` represents a simple path into a YANG schema tree, where some of the list key values may be constrained.

It is encoded in the similar format to the YANG JSON encoding for instance-identifier, section 6.11 of [RFC7951], except with more flexibility on the keys, in that keys may be left-out or be bound to a regular expression filter.

The rules for constructing a YPath are:

- * A YPath is a sequence of data tree path segment separated by a `'/'` character. If the path starts with a `'/'` then it is absolute path from the root of the schema, otherwise it is a relative path, where the context of the relative path must be declared.
- * Constraints on key values may be specified within a single pair of `'[']'` brackets, where:
 - keys may be given in any order, and may be omitted, in which case they match any value. Key matches are separated by a comma `(,)` with optional space character either side.
 - key match is given by `_<key>=<value>_`, with optional space characters either side of the equals `(=)`, and value is specified as:
 - o `'<value>'`, for an exact match of the key's value. Single quote characters `(')` must be escaped with a backslash `(\)`.
 - o `r'<reg-expr>'`, for a regex match of the key value using [RFC9485], and where the regular-expression is a full match of the string, i.e, it implicit anchors to the start and end of the value.

Some examples of YPaths:

- * `_/ietf-interfaces:interfaces/interface[name='eth0']/ietf-ip:ipv6/ip_` - which identifies is `'ipv6/ip'` data node in the `ietf-ip` module for the `'eth0'` interface.
- * `_/ietf-interfaces:interfaces/interface[name=r'eth.*']/ietf-ip:ipv6/ip_` - which identifies all interfaces with a name that start with `"eth"`.

- * `_/example:multi-keys-list[first-key='foo', second-key=r'bar.*']_` - which identifies all entries in the 'multi-keys-list', where the first-key matches foo, and the second-key starts with bar.
- * `_/ietf-interfaces:interfaces/interface_` - which identifies the `_interface_` list data node in the ietf-interfaces module for all interfaces. I.e., the interface list 'name' key is unrestricted.
- * `_/ietf-interfaces:interfaces/interface[]_` - alternative form of the previous YPath.

6.2. The "filters" Container

The "filters" container maintains a list of all datastore subscription filters that persist outside the lifecycle of a single subscription. This enables predefined filters that may be referenced by more than one configured or dynamic subscription.

Below is a tree diagram for the "filters" container. All objects contained in this tree are described in the YANG module in Section 12.

```

module: ietf-yp-lite-config
  +--rw datastore-telemetry!
    +--rw filters
      +--rw filter* [name]
        +--rw name                string
        +--rw (filter)
          +--:(path)
            | +--rw path          ypath
          +--:(subtree)
            | +--rw subtree       <anydata> {ypl:subtree}?
          +--:(xpath)
            | +--rw xpath         yang:xpath1.0 {ypl:xpath}?
          +--rw xpath
            +--rw xpath           yang:xpath1.0 {ypl:xpath}?

  augment /ypl:establish-subscription/ypl:input/ypl:target
    /ypl:filter:
      +--:(by-reference)
  augment /ypl:subscription-started/ypl:target:
    +-- filter-ref?   filter-ref

```

Figure 1

6.3. Decomposing Subscription Filters

In order to address the issues described in Section 3.2, YANG Push Lite allows for publishers to send subtrees of data nodes in separate `_update_` notifications, rather than requiring that the subscription data be returned as a single datastore `_update_` notification covering all data nodes matched by the subscription filter. This better facilitates publishers that internally group some of their operational data fields together into larger structures for efficiency, and avoids publishers or receivers having to consume potentially very large notification messages. For example, each entry in the `_/ietf-interfaces:interface/interface_list` could be represented as an object of data internally within the publisher. In essence, a client specified subscription filter can be decomposed by a publisher into more specific non-overlapping filters that are then used to return the data.

In particular:

1. A Publisher MAY decompose a client specified subscription filter path into a set of non-overlapping subscription filter paths that collectively cover the same data. The publisher is allowed to return data for each of these decomposed subscription filter paths in separate `_update_` notification messages, each with separate, perhaps more precise, `_observation-time_` timestamps, but all using the same notification `_event-time_`.
2. A Publisher MAY split large lists into multiple separate update messages, each with separate `_observation-time_` timestamps, but all using the same notification `_event-time_`. E.g., if a device has 10,000 entries in a list, it may return them in a single response, or it may split them into multiple smaller messages, perhaps for 500 interfaces at a time.

To ensure that clients can reasonably process data returned via decomposed filters then:

1. `_update_` notifications MUST indicate the precise subtree of data that the update message is updating or replacing, i.e., so a receiver can infer that data nodes no longer being notified by the publisher have been deleted:
 - * if we support splitting list entries in multiple updates, then something like a `_more_data_` flag is needed to indicate that the given update message is not complete.

7. Datastore Event Streams

In YANG Push Lite, a subscription, based on the selected filters, will generate a ordered stream of datastore `_update_` records that is referred to as an event stream. Each subscription logically has a different event stream of update records, even if multiple subscriptions use the same filters to select datastore nodes.

As YANG-defined event records are created by a system, they may be assigned to one or more streams. The event record is distributed to a subscription's receiver(s) where (1) a subscription includes the identified stream and (2) subscription filtering does not exclude the event record from that receiver.

Access control permissions may be used to silently exclude event records from an event stream for which the receiver has no read access. See [RFC8341], Section 3.4.6 for an example of how this might be accomplished. Note that per Section 2.7 of this document, subscription state change notifications are never filtered out. *TODO, filtering and NACM filtering should be dependent on whether it is a configured or dynamic subscription.*

If subscriber permissions change during the lifecycle of a subscription and event stream access is no longer permitted, then the subscription MUST be terminated. *TODO, check this*

Event records SHALL be sent to a receiver in the order in which they were generated. I.e., the publisher MUST not reorder the events when enqueueing notifications on the transport session, but there is no guarantee of delivery order.

Event records MUST NOT be sent before a `_subscription-started_` notification (Section 9.2.1) or after a `_subscription-terminated_` notification (Section 9.2.2), or to a particular receiver that has been sent a `_receiver-disconnected_` notification (Section 9.2.3).

7.1. Notification Envelope

All notifications in the event stream MUST be encoded using [I-D.draft-ietf-netconf-notif-envelope] to wrap the notification message, and MUST include the `_event-time_`, `_hostname_`, and `_sequence-number_` leafs in all messages.

The following example illustrates a fully encoded `_update_` notification that includes the notification envelope and additional meta-data fields. The `_update_` notification, i.e., as defined via the `_notification_` statement in the yang-push-lite YANG module, is carried in the `_contents_` anydata data node.

```

{
  "ietf-yp-notification:envelope": {
    "event-time": "2024-10-10T08:00:05.22Z",
    "hostname": "example-router",
    "sequence-number": 3219,
    "contents": {
      "ietf-yp-lite:update": {
        "id": 1011,
        "snapshot-type": "periodic",
        "observation-time": "2024-10-10T08:00:05.11Z",
        "updates": [
          {
            "target-path": "ietf-interfaces:interfaces/interface",
            "data": {
              "ietf-interfaces:interfaces": {
                "ietf-interfaces:interface": [
                  {
                    "name": "eth0",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": true,
                    "ietf-interfaces:oper-status": "up",
                    "ietf-interfaces:admin-status": "up"
                  },
                  {
                    "name": "eth1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": true,
                    "ietf-interfaces:oper-status": "up",
                    "ietf-interfaces:admin-status": "up"
                  }
                ]
              }
            }
          }
        ]
      }
    }
  }
}

```

Figure 2: Example of update notification including notification envelope

7.2. Event Records

A single `_update_` record is used for all datastore notifications. It is used to report the current state of a set of data nodes at a given target path for either periodic, on-change, or resync notifications, and also for on-change notifications to indicate that the data node at the given target path has been deleted.

The schema for this notifications is given in the following tree diagram:

```

+---n update
|   +--ro id?                subscription-id
|   +--ro path-prefix?       string
|   +--ro snapshot-type?     enumeration
|   +--ro observation-time?  yang:date-and-time
|   +--ro updates* [target-path]
|   |   +--ro target-path    string
|   |   +--ro data?          <anydata>
|   +--ro complete?          empty

```

Figure 3: 'update' notification

The normative definitions for the notifications fields are given in the YANG module in Section 12. The fields can be informatively summarized as:

- * `_id_` - identifies the subscription the notification relates to.
- * `_path-prefix_` - identifies the absolute instance-data path to which all target-paths are data are encoded relative to.
- * `_snapshot-type_` - this indicates what type of event causes the update message to be sent. I.e., a periodic collection, an on-change event, or a resync collection.
- * `_observation-time_` - the time that the data was sampled, or when the on-change event occurred that caused the message to be published.
- * `_target-path_` - identifies the data node that is being acted on, either providing the replacement data for, or that data node that is being deleted.
- * `_data_` - the full replacement data subtree for the content at the target-path, encoded from the path-prefix.

- * `_complete_` - if present, this flag indicates that a periodic collection (*TODO, what about on-change*) is complete. Setting this flag is semantically equivalent to the server sending a separate update-complete notification.

As per the structure of the `_update_` notification, a single notification MAY provide updates for multiple target-paths.

7.3. Types of subscription event monitoring

Subscription can either be based on sampling the requested data on a periodic cadence or being notified when the requested data changes. In addition, this specification allows for subscriptions that both notify on-change and also with a periodic cadence, which can help ensure that the system eventually converges on the right state, even if on-change notification were somehow lost or mis-processed anywhere in the data processing pipeline.

The schema for the update-trigger container is given in the following tree diagram:

```

module: ietf-yp-lite-config
  +--rw datastore-telemetry!
    +--rw subscriptions
      +--rw subscription* [name]
        +--rw update-trigger
          +--rw periodic!
            | +--rw period          centiseconds
            | +--rw anchor-time?   yang:date-and-time
          +--rw on-change! {on-change}?
            +--rw sync-on-start?   boolean

  augment /ypl:establish-subscription/ypl:input/ypl:target
    /ypl:filter:
      +--:(by-reference)
  augment /ypl:subscription-started/ypl:target:
    +-- filter-ref?   filter-ref

```

Figure 4: 'update-trigger' container

The normative definitions for the update-trigger fields are given in the `_ietf-yp-lite_` YANG module in Section 12. They are also described in the following sections.

7.4. Periodic events

In a periodic subscription, the data included as part of an update record corresponds to data that could have been read using a retrieval operation. Only the state that exists in the system at the time that it is being read is reported, periodic updates never explicitly indicate whether any data-nodes or list entries have been deleted. Instead, receivers must infer deletions by the absence of data during a particular collection event.

For periodic subscriptions, triggered updates will occur at the boundaries of a specified time interval. These boundaries can be calculated from the periodic parameters:

- * a `_period_` that defines the duration between push updates.
- * an `_anchor-time_`; update intervals fall on the points in time that are a multiple of a `_period_` from an `_anchor-time_`. If an `_anchor-time_` is not provided, then the publisher chooses a suitable anchor-time, e.g., perhaps the time that the subscription was first instantiated by the publisher.

The anchor time and period are particularly useful, in fact required, for when the collected telemetry data is being stored in a time-series database and the subscription is setup to ensure that each collection is placed in a separate time-interval bucket.

Periodic update notifications are expected, but not required, to use a single `_target-path_` per `_update_` notification.

7.5. On-Change events

In an on-change subscription, `_update_` records indicate updated values or when a monitored data node or list node has been deleted. An `_update_` record is sent whenever a change in the subscribed information is detected. `_update_` records SHOULD be generated at the same subtree as equivalent periodic subscription rather than only the specific data node that is on-change notifiable. The goal is to ensure that the `_update_` message contains a consistent set of data on the subscription path.

Each entry in the `_updates_` list identifies a data node (i.e., list entry, container, leaf or leaf-list), via the `_target-path_` that either has changes in state or has been deleted.

A delete of a specific individual data node or subtree may be notified in two different ways:

- * if the data that is being deleted is below the `_target-path_` then the delete is implicit by the publisher returning the current data node subtree with the delete data nodes missing. I.e., the receiver must implicitly infer deletion.
- * if the data node is being deleted at the target path. E.g., if an interface is deleted then an entire list entry related to that interface may be removed. In this case, the `_target path_` identifies the list entry that is being deleted, but the data returned is just an empty object {}, which replaces all the existing data for that object in the receiver. *TODO, is this better as a delete flag, or separate delete list?*

On-change subscriptions also support the following additional parameters:

- * `_sync-on-start_` defines whether or not a complete snapshot of all subscribed data is sent at the start of a subscription. Such early synchronization establishes the frame of reference for subsequent updates.

7.5.1. On-Change Notifiable Datastore Nodes

Publishers are not required to support on-change notifications for all data nodes, and they may not be able to generate on-change updates for some data nodes. Possible reasons for this include:

- * the value of the datastore node changes frequently (e.g., the `in-octets` counter as defined in [RFC8343]),
- * small object changes that are frequent and meaningless (e.g., a temperature gauge changing 0.1 degrees),
- * or no implementation is available to generate a notification when the source variable for a particular data node has changed.

In addition, publishers are not required to notify every change or value for an on-change monitored data node. Instead, publishers MAY limit the rate at which changes are reported for a given data node, i.e., effectively deciding the interval at which an underlying value is sampled. If a data node changes value and then reverts back to the original value within a sample interval then the publisher MAY not detect the change and it would go unreported. However, if the data node changes to a new value after it has been sampled, then the change and latest state are reported to the receiver. In addition, if a client was to query the value (e.g., through a NETCONF `get-data` RPC) then they MUST see the same observed value as would be notified.

To give an example, if the interface link state reported by hardware is changing state hundreds of times per second, then it would be entirely reasonable to limit those interface state changes to a much lower cadence, e.g., perhaps every 100 milliseconds. In the particular case of interfaces, there may also be data model specific forms of more advanced dampening that are more appropriate, e.g., that notify interface down events immediately, but rate limit how quickly the interface is allowed to transition to up state, which overall acts as a limit on the rate at which the interface state may change, and hence also act as a limit on the rate at which on-change notifications could be generated.

The information about what nodes support on-change notifications is reported using capabilities operational data model. This is further described in Section 11.

7.5.2. On-Change Considerations

On-change subscriptions allow receivers to receive updates whenever changes to targeted objects occur. As such, on-change subscriptions are particularly effective for data that changes infrequently but for which applications need to be quickly notified, with minimal delay, whenever a change does occur.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

Whether or not to accept or reject on-change subscription requests when the scope of the subscription contains objects for which on-change is not supported is up to the publisher implementation. A publisher MAY accept an on-change subscription even when the scope of the subscription contains objects for which on-change is not supported. In that case, updates are sent only for those objects within the scope of the subscription that do support on-change updates, whereas other objects are excluded from update records, even if their values change. In order for a subscriber to determine whether objects support on-change subscriptions, objects are marked accordingly on a publisher. Accordingly, when subscribing, it is the responsibility of the subscriber to ensure that it is aware of which objects support on-change and which do not. For more on how objects are so marked, see Section 3.10. *TODO Is this paragraph and the one below still the right choice for YANG Push Lite?*

Alternatively, a publisher MAY decide to simply reject an on-change subscription if the scope of the subscription contains objects for which on-change is not supported. In the case of a configured subscription, the publisher MAY suspend the subscription.

7.6. Combined periodic and on-change subscriptions

A single subscription may be created to generate notifications both when changes occur and on a periodic cadence. Such subscriptions are equivalent to having separate periodic and on-change subscriptions on the same path, except that they share the same subscription-id and filter paths.

7.7. Streaming Update Examples

TODO, Generate new JSON based example of a periodic, and delete messages. Current placeholders are the existing YANG Push Notifications.

Figure XXX provides an example of a notification message for a subscription tracking the operational status of a single Ethernet interface (per [RFC8343]). This notification message is encoded XML _W3C.REC-xml-20081126_ over the Network Configuration Protocol (NETCONF) as per [RFC8640].

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
<eventTime>2017-10-25T08:00:11.22Z</eventTime>
<push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
  <id>1011</id>
  <datastore-contents>
    <interfaces
      xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>eth0</name>
        <oper-status>up</oper-status>
      </interface>
    </interfaces>
  </datastore-contents>
</push-update>
</notification>
```

Figure 5: Example 'update' periodic notification

Figure XXX provides an example of an on-change notification message for the same subscription.

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
<eventTime>2017-10-25T08:22:33.44Z</eventTime>
<push-change-update
  xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
  <id>89</id>
  <datastore-changes>
    <yang-patch>
      <patch-id>0</patch-id>
      <edit>
        <edit-id>edit1</edit-id>
        <operation>replace</operation>
        <target>/ietf-interfaces:interfaces</target>
        <value>
          <interfaces
            xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
              <interface>
                <name>eth0</name>
                <oper-status>down</oper-status>
              </interface>
            </interfaces>
          </value>
        </edit>
      </yang-patch>
    </datastore-changes>
  </push-change-update>
</notification>
```

Figure 6: Example 'update' on-change notification

8. Receivers, Transports, and Encodings

8.1. Receivers

Every subscription is associated with one or more receivers, which each identify the destination host, transport and encoding settings, where all notifications for a subscription are sent.

For configured subscriptions there is no explicit association with an existing transport session, and hence the properties associated with the receiver are explicitly configured, as described in Section 8.1.1.

For dynamic subscriptions, the receiver, and most associated properties are implicit from the session on which the dynamic subscription was initiated, as described in Section 8.1.2.

8.1.1.1. Receivers for Configured Subscriptions

For configured subscriptions, receivers are configured independently from the subscriptions and then referenced from the subscription configuration.

Configured subscriptions MAY have multiple receivers, e.g., to facilitate redundancy. Each receiver MAY be configured with different transports and associated transport settings, but they MUST all share the same encoding.

All subscription notifications, including lifecycle notifications (Section 9.2), are sent to all receivers except for the `_receiver-disconnected_` notification, which is only sent to the affected receiver, and only if the subscription remains active because of other active receivers.

Below is a tree diagram for `_datastore-telemetry/receivers_` container. All objects contained in this tree are described in the YANG module in Section 12.2.

These parameters identify how to connect to each receiver. For each subscription, the publisher uses the referenced receiver configuration to establish transport connectivity to the receiver.

```

module: ietf-yp-lite-config
  +--rw datastore-telemetry!
    +--rw receivers
      +--rw receiver* [name]
        +--rw name                string
        +--rw encoding            ypl:encoding
        +--rw dscp?              inet:dscp
        +---x reset
        +--rw (notification-message-origin)?
          | +--:(interface-originated)
          | | +--rw source-interface?  if:interface-ref
          | | | {interface-designation}?
          | +--:(address-originated)
          | | +--rw source-vrf?        leafref {supports-vrf}?
          | | +--rw source-address?    inet:ip-address-no-zone
          +--rw (transport-type)
        augment /ypl:establish-subscription/ypl:input/ypl:target
          /ypl:filter:
            +--:(by-reference)
        augment /ypl:subscription-started/ypl:target:
          +-- filter-ref?  filter-ref

```

Figure 7: datastore-telemetry/receivers container

Each configured receiver has the following associated properties:

- * a `_name_` to identify and reference the receiver in the subscription configuration.
- * a `_transport_`, which identifies the transport protocol to use for all connections to the receiver.
 - any transport-specific related parameters, some of which may be mandatory, others optional to specify, e.g., DSCP. There are likely to be various data nodes related to establishing appropriate security and encryption.
- * an `_encoding_` to encode all YANG notification messages to the receiver, i.e., see Section 8.3.
- * optional parameters to identify where traffic should egress the publisher:
 - a `_source-interface_`, identifying the egress interface to use from the publisher, implicitly choosing the source IP address and VRF.
 - a `_source-vrf_`, identifying the Virtual Routing and Forwarding (VRF) instance on which to reach receivers. This VRF is a network instance as defined in [RFC8529]. Publisher support for VRFs is optional and advertised using the `_supports-vrf_` feature.
 - a `_source-address_` address, identifying the IP address to source notification messages from.

If none of the above parameters are set, the publisher MAY choose which interface(s) and address(es) to source subscription notifications from.

This specification is transport independent, e.g., see Section 8.2, and thus the YANG module defined in Section 12.2 cannot directly define and expose these transport parameters. Instead, receiver-specific transport connectivity parameters MUST be configured via transport-specific augmentations to the YANG choice node `_/datastore-telemetry/receivers/receiver/transport-type_`.

A publisher supporting configured subscriptions clearly must support at least one YANG data model that augments transport connectivity parameters onto `_/datastore-telemetry/receivers/receiver/transport-`

type_. For an example of a similar such augmentation (but for YANG Push), see [I-D.draft-ietf-netconf-udp-notif]. *TODO, this reference and text will need to be updated to a UDP-notif bis document, that augments the new YANG Push Lite receiver path.*

8.1.2. Receivers for Dynamic Subscriptions

For dynamic subscriptions, each subscription has a single receiver that is implicit from the host that initiated the _establish-subscription_ RPC, reusing the same transport session for all the subscription notifications.

Hence most receiver parameters for a dynamic subscription, e.g., related to the transport, are implicitly determined and cannot be explicitly controlled.

Dynamic subscriptions MUST specify an encoding (see Section 8.3) and MAY specify DSCP Marking (see Section 8.2.1.1) for the telemetry notifications in the _establish-subscription_ RPC (see Figure 14).

8.1.3. Receiver Session States and State Machine

Each subscription will need to establish a subscription to the specified receiver. Multiple subscriptions may share one or more transport sessions to the same receiver.

A receiver in YANG Push Lite can be in one of the following states:

- * ***Configured***: The receiver has been configured on the publisher, but the receiver is not referenced by any valid subscriptions and hence there is no attempt to establish a connection to the receiver.
- * ***Connecting***: The receiver has at least one associated subscription and the publisher is attempting to establish a transport session and complete any required security exchanges, but this process has not yet succeeded.
- * ***Active***: The receiver has at least one associated subscription, a transport session has been established (if required), security exchanges have successfully completed, and the publisher is able to send notifications to the receiver.

The state transitions for a receiver are illustrated below:

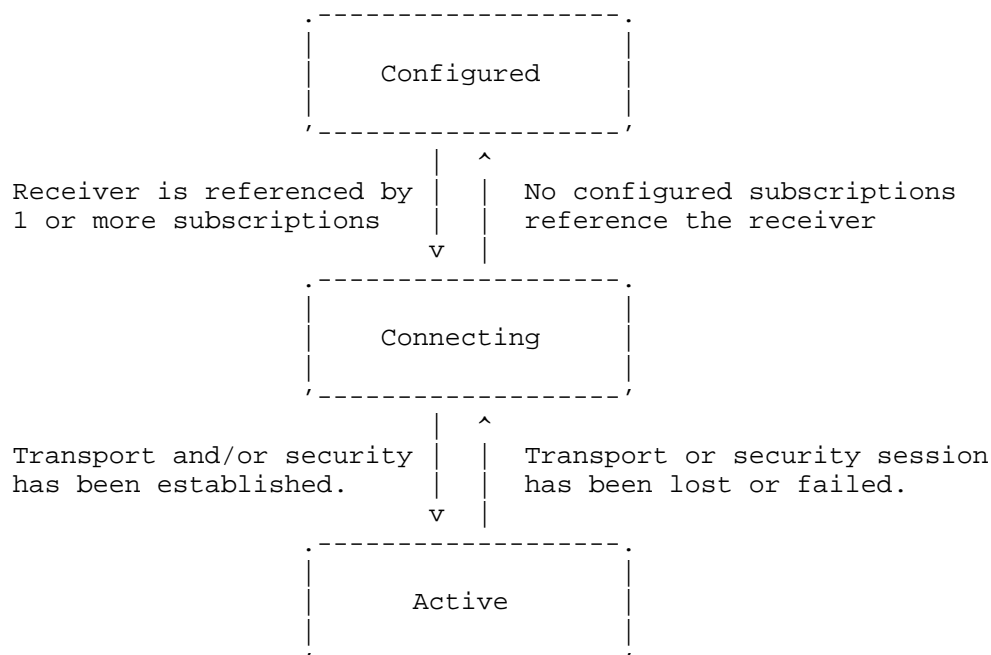


Figure 8: Receiver Session State Diagram

This state model allows implementations and operators to clearly distinguish between receivers that are simply configured, those that are in the process of connecting, and those that are actively being used.

If the configuration has changed such that there were previously connections to a receiver, but that receiver is no longer referenced by valid subscriptions, then the publisher **MUST** close any associated transport sessions to the receiver, but **MAY** delay the closing for a short period of time (no more than 15 minutes) to potentially allow existing transport session to be reused by new subscriptions.

8.2. Transports

This document describes a transport-agnostic mechanism for subscribing to YANG datastore telemetry. Hence, separate specifications are required to define transports that support YANG Push Lite. The requirements for these transport specifications are documented in the following section:

8.2.1. Requirements for Yang Push Lite Transport Specifications

This section provides requirements for any transport specifications supporting the YANG Push Lite solution presented in this document.

The transport specification **MUST** provide YANG modules, to be implemented by publishers implementing the YANG Push configuration model in Section 13, that:

- * augments the `_datastore-telemetry/receivers/transport-type_` choice statement with a container that both identifies the transport and contains all transport specific parameters.
- * augments `_/sysc:system-capabilities/transport/transport-capabilities/_` container with any transport specific capabilities or options (conditional on a YANG `_when_` statement). Note, encodings for a given transport are advertised directly via the `ietf-yp-lite-capabilities` YANG Model Section 14.2.

Using a secure transport is RECOMMENDED. Thus, any transport specification **MUST** provide a mechanism to ensure secure communication between the publisher and receiver in a hostile environment, e.g., through the use of transport layer encryption. Transport specifications **MAY** also specify a mechanism for unencrypted communications, which can be used when transport layer security is not required, e.g., if the transport session is being secured via another mechanism, or when operating within a controlled environment or test lab.

Any transport specification **SHOULD** support mutual receiver and publisher authentication at the transport layer.

The transport selected by the subscriber to reach the publisher **SHOULD** be able to support multiple "establish-subscription" requests made in the same transport session.

The transport specification **MUST** specifying how multiple subscriptions referencing the same receiver are to be handled at the transport layer. The transport specification **MAY** require separate transport sessions per subscription to a given receiver, or it **MAY** allow multiple subscriptions to the same receiver to be multiplexed over a shared transport session.

A specification for a transport built upon this document can choose whether to use the same logical channel for the RPCs and the event records. However, the `_update_` records and the subscription state change notifications **MUST** be sent on the same transport session.

The transport specification MAY specify a keepalive mechanism to keep the transport session alive. There is no YANG Push Lite protocol or application level keepalive mechanism.

TODO, do we need to mention anything about transport session timeouts, e.g., which would cause a subscription to be terminated. What about buffering? Is that a transport consideration?

Additional transport requirements may be dictated by the choice of transport used with a subscription.

8.2.1.1. DSCP Marking

YANG Push Lite supports `_dscp_` marking to differentiate prioritization of notification messages during network transit.

A receiver with a `_dscp_` leaf results in a corresponding Differentiated Services Code Point (DSCP) marking [RFC2474] being placed in the IP header of any resulting `_update_` notification messages and subscription state change notifications. A publisher MUST respect the DSCP markings for subscription traffic egressing that publisher.

The transport specification MUST specify if there are any particular quality-of-service or class-of-service considerations related to handling DSCP settings associated with the subscription.

8.3. Encodings

The `_update_` notification (Section 7.2) and subscription lifecycle notifications (Section 9.2) can be encoded in any format that has a definition for encoding YANG data. For a given subscription, all notification messages are encoded using the same encoding.

Some IETF standards for YANG encodings known at the time of publication are:

- * JSON, defined in [RFC7951]
- * CBOR, defined in [RFC9254], and [RFC9595] for using compressed schema identifiers (YANG SIDs)
- * XML, defined in [RFC7950]

To maximize interoperability, all implementations are RECOMMENDED to support both JSON and CBOR encodings. Constrained platforms may not be able to support JSON and hence may choose to only support CBOR encoding. JSON encoding may not be supported in the scenario that

another encoding becomes the defacto standard (e.g., as JSON has largely replaced XML as the defacto choice for text based encoding). Support for the XML encoding and/or CBOR encoding using YANG SIDs is OPTIONAL.

Encodings are defined in the `_ietf-yp-lite.yang` as YANG identities that derive from the `_encoding_` base identity. Additional encodings can be defined by defining and implementing new identities that derive from the `_encoding_` base identity, and also advertising those identities as part of the `ietf-yp-lite-capabilities` YANG module's transport capabilities Section 14.2.

9. Setting up and Managing Subscriptions

Subscriptions can be set up and managed in two ways:

1. Configured Subscriptions - a subscription created and principally controlled by configuration.
2. Dynamic Subscriptions - a subscription created and principally controlled via YANG RPCs from a telemetry receiver.

Conformant implementations MUST implement at least one of the two mechanisms above for establishing and maintaining subscriptions, but they MAY choose to only implement a single mechanism.

The core behavior for both configured and dynamic subscription is the same, with the key differentiation being how they are provisioned, and how the transport is setup. This next section describes the functionality that is common to both types of subscription, followed by the sections that describe the specifics and differences between the two ways of managing subscriptions.

9.1. Common Subscription Parameters

All subscriptions require the following state to be instantiated:

- * a `_name_` to identify the subscription.
- * the `_target_` for the subscription, comprising:
 - the target datastore, as per [RFC8342]
 - a set of selection filters to choose which datastore nodes the subscription is monitoring or sampling, as described in Section 6.

- * the `_update-trigger_` to indicate when `_update_` notifications are generated:
 - `_periodic_`, for the publisher to send updated copies of the state on a periodic basis
 - `_on-change_`, for the publisher to send state updates when the internal state changes, i.e., event driven.
- * receiver, transport, and encoding parameters, as per Section 8.1. How these are provided differs for configured vs dynamic subscriptions and is further explained in the sections below.

Subscription names MUST be unique across all configured and dynamic subscriptions. Configured subscription take precedence over dynamic subscription, so:

- * attempts to create a dynamic subscription with a name that conflicts with any other subscription (configured or dynamic) MUST fail,
- * configuring a subscription, assuming it passes configuration validation, replaces any dynamic subscriptions with the same name. Thus, causing the dynamic subscription to be immediately terminated (see Section 9.1.4).
- * *TODO: We want to make the name optional to be specified for dynamic subscriptions? If not provided then the publisher could allocate using "dyn-`<XXX>`", where XXX is an increasing numeric subscription id?*

9.1.1. Subscription States

YANG Push Lite has a small set of simple states for a subscription on a publisher. These states are intended to help clients easily determine the health and status of a subscription.

- * ***Invalid***: a subscription that is invalid for any reason. E.g., the subscription references an invalid filter expression for the current device schema. Normally, invalid configurations should be rejected by the system, whether due to subscription configuration or `_establish-subscription_` RPC, and hence this state should rarely be seen.
- * ***Inactive***: a valid subscription, but one that is not active because it has no associated receivers. This state is unlikely to be seen for dynamic subscriptions.

- * ***Connecting***: a subscription that is valid, and has appropriate receiver configuration, but the publisher has not managed to successfully connect to any receivers yet, and hence has not sent a `_subscription-started_` notification. Transport security failures would be in this state. ***TODO**, what about no route to the receiver?*
- * ***Active***: a valid subscription, connected to all specified receivers, that has sent a `_subscription-stated_` notification and is generating `_update_` notifications, as per the terms of the subscription update policy.
- * ***Partial***: this is the same as per the description of `_Active_` above except that it has only managed to successfully connect to some receivers, not all of specified receivers associated with the subscription.
- * ***Terminated***: represents a subscription that has finished. Subscriptions would only be expected to transiently be in this state and hence it would not normally be reported. Terminated dynamic subscriptions, or unconfigured subscriptions, should quickly be removed from the operational state of the device. Terminated

Below is a state diagram illustrating the states, and the likely changes between states. However, this is not a formal state machine and publishers can move between arbitrary states based on changes to subscription properties, the system, connectivity to receivers, or resource constraints on the system. New subscriptions should choose an appropriate starting state, e.g., either `Inactive` or `Invalid`.

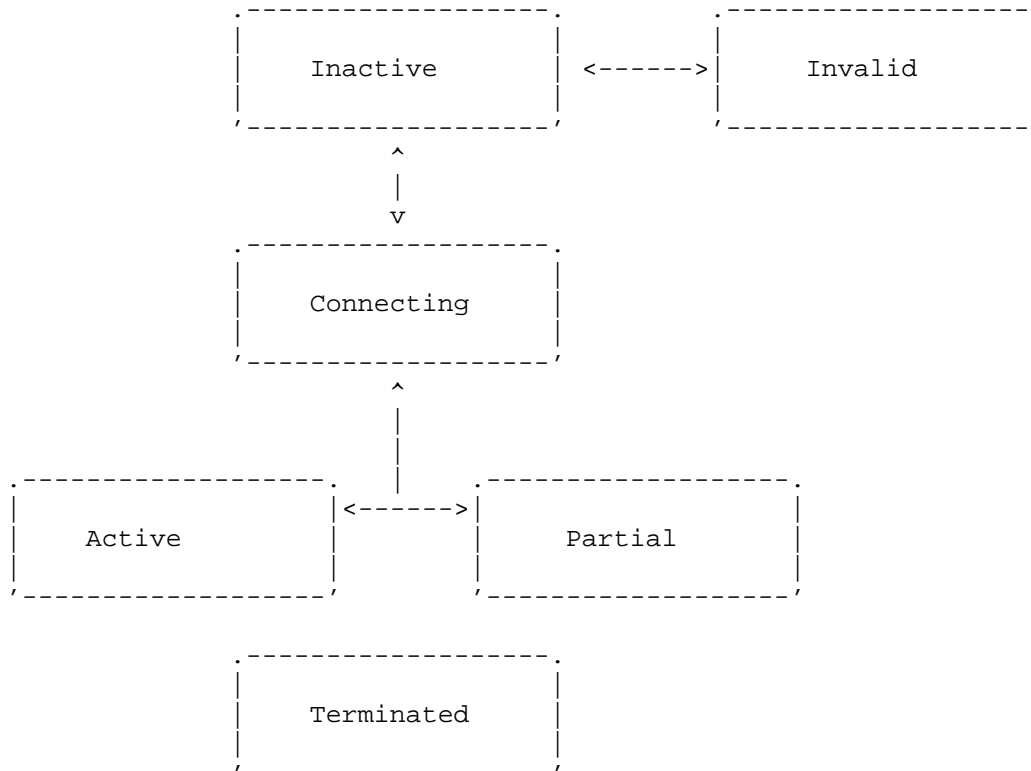


Figure 9: Publisher's States for a Subscription

9.1.2. Creating Subscriptions

After a subscription is successfully established, the publisher immediately sends a `_subscription-started_` subscription state change notification to each receiver. It is quite possible that upon configuration, reboot, or even steady-state operations, a transport session may not be currently available to the receiver. In this case, when there is something to transport for an active subscription, transport-specific "call home" operations [RFC8071] will be used to establish the connection. *TODO - this should just reference the transport RFCs*. When transport connectivity is available, notification messages may then be pushed.

With active configured subscriptions, it is allowable to buffer event records even after a `_subscription-started_` has been sent. However, if events are lost (rather than just delayed) due to buffer capacity being reached, a `_subscription-terminated_` notification **MUST** be sent, followed by a new `_subscription-started_` notification. These notifications indicate an event record discontinuity has occurred.

9.1.3. Modifying Subscriptions

Subscriptions **MAY** be modified in various ways:

1. the subscription `_filter_` (inline, referring to a different named filter, or changing the referenced filter), `_update-triggers_`, or `_description_` may be changed.
2. `_receivers_` may be added or removed.
3. parameter associated with a receiver may be changed.

The following sections describe how these changes are handled.

9.1.3.1. Changing configuration associated with the subscription

If any of the following properties associated with a subscription are changed:

- * the content of an inline filter,
- * the name, or content, of a referenced filter,
- * the `_encoding_`,
- * the `_update-triggers_`,
- * the `_description_` field,
- * the YANG schema used by subscription,
- * any other fields that are included in a `_subscription-started_` notification message

then the subscription must be terminated, as per Section 9.1.4, and then recreated, as per Section 9.1.2. This can be used to trigger any necessary actions on the receiver to ensure that it can successfully process any changes to the subscription `_update_` notifications.

If the publisher believes that the schema associated with a subscription may have changed then it may defensively terminate and recreate the subscription.

9.1.3.2. Adding receivers

If the modification involves adding one or more receivers, then once the transport session to the new receiver(s) has been established, i.e., they have reached the `_receiver active_` state (Section 8.1.3), then a new `_subscription-started_` notification (Section 9.2.1) MUST be sent on the subscription to `*all receivers*`.

9.1.3.3. Removing receivers

If all receivers, or the last active receiver, is removed then the subscription becomes `_inactive_` and a `_subscription-terminated_` notification SHOULD be sent to all receivers, if possible, prior to bringing down any session to the receivers. Obviously, if the transport session to the receiver has closed then it will not be possible to send a `_subscription-terminated_` notification to that receiver.

If one or more receivers are removed, but at least one receiver for the subscription remains active, then a `_receiver-disconnected_` notification is sent `_only_` to the receivers that has been removed from the subscription. Any receivers that are still active don't receive any notification.

9.1.3.4. Changing receiver configuration

If the settings associated with a receiver are changed, then the publisher has two choices:

1. It MAY treat the change as a removal of the receiver (Section 9.1.3.3), followed by the addition of a receiver (Section 9.1.3.2).
2. If it is a minor configuration change, e.g., a change to the DSCP settings, then it MAY dynamically update the transport session behavior without notifying the receiver.

TODO, should we add changing the source address/port as an example here? Or it is just simpler to just always force the receiver to be removed and re-added for simplicity? Is further prose required about what is allowed to be handled in this fashion?

9.1.4. Terminating Subscriptions

Subscriptions can be terminated by the publisher due to any of the following circumstances:

1. The subscription has been unconfigured.
2. Some subscription, receiver, transport or encoding configuration has been removed, e.g., receiver configuration, such that there is no longer the sufficient minimum information to maintain the subscription.
3. A dynamic subscription has been terminated via a `_delete-subscription_` or `_kill-subscription_` YANG RPC.
4. Transport connectivity to all receivers have been lost, either due to network issues, or a failure in the security session state.
5. The publisher doesn't have sufficient resources to honor the terms of the subscription, i.e., it is generating too many `_update_` notifications, or attempting to send too much data.
6. The subscription parameters have changed in such a way that means the subscription needs to be reset by terminating and recreating it, i.e., as described for some changes in Section 9.1.3.1.
7. The `_reset_` RPC is invoked on a configured subscription, or on the referenced receiver associated with a configured subscription.
8. From a receiver perspective, if transport connectivity is lost, then that is equivalent to terminating the subscription via a `_subscription-terminated_` notification.

TODO, check this list, some of these are not really MAYs, but MUSTs, but perhaps those are the references into this section.

If possible, the publisher MUST try and close the subscription gracefully by generating a `_subscription-terminated_` notification to all receivers before closing any sessions to any receivers that have no remaining subscriptions. Publishers MAY complete their current collection if one is in progress before generating the `_subscription-terminated_` notification. Obviously, if transport connectivity to a receiver has been lost then neither of these two actions will be possible.

The publisher MUST NOT generate any further events, e.g., `_update_` notifications, related to the subscription after the `_subscription-terminated_` notification has been generated. In addition, receivers SHOULD ignore any messages received outside of an active subscription, i.e., either before a `_subscription-started_` notification or after a `_subscription-terminated_` or `_receiver-disconnected_` notification.

If the publisher accepts the request, which it MUST, if the `subscription-id` matches a dynamic subscription established in the same transport session, then it should stop the subscription and send a `_subscription-terminated_` notification.

9.2. Subscription Lifecycle Notifications

In addition to sending event records to receivers, a publisher also sends subscription lifecycle state change notifications when lifecycle events related to subscription management occur.

Subscription state change notifications are generated per subscription, and are injected into the stream of `_update_` messages for that that subscription. These notifications MUST NOT be dropped or filtered.

Future extensions, or implementations MAY augment additional fields into the notification structures. Receivers MUST silently ignore unexpected fields.

The complete set of subscription state change notifications is described in the following subsections:

9.2.1. "subscription-started"

The subscription started notification is sent to a receiver to indicate that a subscription is active and they may start to receive `_update_` records from the publisher.

The subscription started notification may be sent for any of these reasons:

1. A new subscription (configured or dynamic) has been started.
2. A receiver has been added to a subscription.
3. The properties of a subscription has been changed, in which case a `_subscription-terminated_` notification should be sent, followed by a `_subscription-started_` notification if the new properties are valid.

4. A configured subscription previously failed, and was terminated. After the publisher has successfully re-established a connection to the receiver and is read to send datastore event records again.

Below is the tree diagram for the `_subscription-started_` notification. All data nodes contained in this tree diagram are described in the YANG module in Section 12.2.

```

+---n subscription-started
|   +--ro id                subscription-id
|   +--ro name              subscription-name
|   +--ro description?      string
|   +--ro target
|   |   +--ro datastore?    identityref
|   |   +--ro (filter)
|   |   |   +---:(path)
|   |   |   |   +--ro path      ypath
|   |   |   +---:(subtree)
|   |   |   |   +--ro subtree   <anydata> {ypl:subtree}?
|   |   |   +---:(xpath)
|   |   |       +--ro xpath     yang:xpath1.0 {ypl:xpath}?
|   +--ro update-trigger
|   |   +--ro periodic!
|   |   |   +--ro period        centiseconds
|   |   |   +--ro anchor-time?  yang:date-and-time
|   |   +--ro on-change! {on-change}?
|   |   +--ro sync-on-start?    boolean

```

Figure 10: subscription-started Notification Tree Diagram

9.2.2. "subscription-terminated"

For a receiver, this notification indicates that no further event records for an active subscription should be expected from the publisher unless and until a new `_subscription-started_` notification is received.

A `_subscription-terminated_` notification SHOULD only be sent by a publisher to a receiver if a `_subscription-started_` notification was previously sent.

The subscription terminated notification may be sent to a receiver for any of these reasons:

1. A receiver has been removed from a configured subscription.

2. A subscription has been stopped, either due to the change/removal of some configuration, or an RPC has been used to delete or kill a subscription.
3. The properties of a subscription has been changed, in which case a `_subscription-terminated_` notification should be sent, followed by a `_subscription-started_` notification if the new properties are valid.
4. A subscription has failed for any reason, e.g.,:
 - * The publisher is no longer able to honor the subscription, due to resource constraints, or the filter is no longer valid.
 - * Any transport level buffer to the receiver has become full, and the hence the publisher is dropping `_update_` notifications.

Below is a tree diagram for "subscription-terminated". All objects contained in this tree are described in the YANG module in Section 12.2.

```

+---n subscription-terminated
|   +--ro name          subscription-name
|   +--ro id            subscription-id
|   +--ro reason        identityref

```

Figure 11: subscription-terminated Notification Tree Diagram

TODO Augmenting extra fields is better for clients? The `_reason_` data node `identityref` indicates why a subscription has been terminated, and could be extended with further reasons in future. I suggest that we change this to an enum with an optional description field.**

9.2.3. "receiver-disconnected"

Configured subscriptions support multiple receivers. If one particular receiver is slow to process notifications, e.g., causing

The `_receiver-disconnected_` notification is sent to a specific receiver if the publisher has a multi-receiver subscription, and the publisher has decided to stop sending notification to a particular receiver, but the subscription will continue sending updates to other receivers. Note the `_subscription-terminated_` notification is a publisher is stopping sending messages to all receivers.

Possible reasons as to why a publisher may disconnect a receiver are:

- * the receiver is too slow processing the messages causing the subscription to back pressure overflowing internal buffers,
- * the transport session is unreliable, frequently dropping and reconnecting.

```

+---n update-complete
  +--ro id      subscription-id

```

Figure 12: update-complete Notification Tree Diagram

9.2.4. "update-completed"

TODO, this description needs to be updated.

This notification indicates that all of the event records prior to the current time have been passed to a receiver. It is sent before any notification messages containing an event record with a timestamp later than (1) the subscription's start time.

After the `_update-complete_` notification has been sent, additional event records will be sent in sequence as they arise naturally on the publisher.

Below is a tree diagram for `_update-complete_`. All objects contained in this tree are described in the YANG module in Section 4.

```

+---n update-complete
  +--ro id      subscription-id

```

Figure 13: update-complete Notification Tree Diagram

9.3. Configured Subscriptions

Configured subscriptions allow the management of subscriptions via configuration so that a publisher can send notification messages to a receiver.

This document specifies the `ietf-yp-lite-config` YANG module Section 13.2 that defines an configuration model for configuring subscriptions. Support for this YANG module is OPTIONAL and is advertised using the normal YANG mechanisms, e.g., [RFC8525]. *TODO, do we also advertise support via capabilities, i.e., issue 16 (<https://github.com/rgwilton/draft-yp-observability/issues/16>)*

In addition to the common subscription parameters described in Section 9.1, a configured subscription also includes:

- * one or more associated receivers for the subscription, as described in Section 8.1. The references receivers specify all transport, receiver, and encoding parameters.

Configured subscriptions have several characteristics distinguishing them from dynamic subscriptions, such as:

- * configured subscriptions are created, modified or deleted, by any configuration client with write permission on the subscription configuration.
- * configured subscriptions have the ability to send notification messages to more than one receiver. All receivers for a given subscription MUST use the same encoding.
- * The lifetime of a configured subscription is tied to the configuration. I.e., if a valid and complete configuration exists for a subscription, then the publisher MUST attempt to connect to the receiver and honor the requirements of the subscription. In particular:
 - If the configuration is altered or removed then the subscription will similarly be altered or removed.
 - If the device reboots, then the configured subscription will obviously end, but once the subscription configuration has been processed after boot up, then the subscription will be recreated again, assuming the subscription configuration is still valid.
 - If transport connectivity to the receiver is broken, then any subscriptions using that transport are terminated, but the publisher MUST periodically attempt to re-establish connection to the receiver and re-activate any configured subscriptions to that receiver.
 - Note, if there are no active subscriptions for a given receiver then any transport session(s) associated with that receiver MUST be closed, but that MAY be after a short delay.
- * Other than the `_reset_` YANG Action, described in Section 9.3.4, there are no YANG RPCs to dynamically create, modify, or delete a configured subscription, any alterations MUST be done via changes to the configuration.

9.3.1. Creating a Configured Subscription

Configured subscriptions are those created via changes to the publisher's configuration, e.g., using the YANG module defined in Section 13, or an equivalent configuration mechanism, such as a command-line interface, or alternative YANG configuration model.

After the configuration change has been accepted by the system, then the subscription is updated, as per Section 9.1.2.

TODO, to see an example of subscription creation using configuration operations over NETCONF, see Appendix A.

9.3.2. Modifying a Configured Subscription

Configured subscriptions can be modified due to configuration changes in the subscription configuration or referenced configuration, i.e., filters or receivers. After the configuration change has been accepted by the system, then the subscription is updated, as per Section 9.1.3.

9.3.3. Deleting a Configured Subscription

Configured subscriptions can be deleted via configuration. After the configuration change has been accepted by the system the subscription is terminated, as per Section 9.1.4.

9.3.4. Resetting a Configured Subscription

Configured subscriptions are generally expected to self-monitor and automatically reconnect to the receivers if they experience network or transport issues. However, the data model also defines explicit `YANG _actions_` to either: (i) reset a single subscription, or (ii) reset all subscriptions and the transports(s) associated with a specific configured receiver instance.

These reset actions primarily act at the subscription application layer, but may be useful if a receiver or collector determines that a configured subscription is not behaving correctly, and wishes to force a reset of the subscription without modifying the configuration associated with the subscription or forcing a configuration change on the publisher device.

The reset action on a subscription is handled equivalently to removing and re-adding the subscription configuration. I.e., the subscription MUST be terminated, as per Section 9.1.4 before being recreated, as per Section 9.1.2. The reset action also resets (terminated and re-establishes) any subscription specific transport session that is not shared with any other subscriptions.

The reset action on a receiver is handled equivalently to removing and re-adding the receiver configuration for the receiver that has been reset. Specifically, every subscription referencing the receiver MUST be terminated, as per Section 9.1.4 before being recreated, as per Section 9.1.2. Any transport sessions tied to the subscriptions referencing the reset receiver MUST also be terminated and re-established.

9.4. Dynamic Subscriptions

Dynamic subscriptions are where a subscriber initiates a subscription negotiation with a publisher via a YANG RPC [RFC7950].

Support for dynamic subscriptions is OPTIONAL, with its availability advertised via the `_dynamic_` YANG feature in the `ietf-yp-lite` YANG module Section 12.2, and also via the capabilities module Section 14.2.

Dynamic subscription differ from configured subscription in the following ways:

- * Dynamic subscription reuse the same transport session on which the `_establish-subscription_` RPC was received to send back any notifications, and so the transport and receiver do not need to be specified and each dynamic subscription can always only have a single receiver.
- * The publisher MUST reply to the `_establish-subscription_` RPC before sending the `_subscription-started_` or any `_update_` notifications for this subscription.
- * The lifetime of a dynamic subscription is bound by the transport session used to establish it. If the transport session fails then the dynamic subscription MUST be terminated.
- * Active dynamic subscriptions cannot be modified on the fly, instead, the client is required to send a `_delete-subscription_` YANG RPC followed by a `_establish-subscription_` YANG RPC.

- * Dynamic subscriptions can either be terminated by the client that established the subscription sending a `_delete-subscription_` YANG RPC on the same transport session, or any client with sufficient access permissions invoking the `_kill-subscription_` YANG RPC.
- * A publisher MAY terminate a dynamic subscription at any time, i.e., due to internal constraints of the publisher.
- * If a dynamic subscription is terminated for any reason, then the client is responsible for re-establishing the subscription if it is still required.
- * If the publisher cannot honor the terms of a dynamic subscription then the subscription MUST be terminated. *TODO, is this a SHOULD or MUST, do we want some leeway for temporary issues? E.g., allow some buffering. Also, this effectively applies to config subscriptions as well and hence should move.*

9.4.1. Establishing a Dynamic Subscription

The `_establish-subscription_` RPC allows a subscriber to request the creation of a subscription.

In addition to the common subscription parameters described in Section 9.1, the `_establish-subscription_` YANG RPC:

- * includes the `_encoding_` to be used for all YANG push lite notifications
- * optionally includes DSCP settings to use for the transport.
- * a dynamic subscription may reference a configured target `_filter_`. If the configuration for the referenced filter changes then the subscription MUST be terminated Section 9.1.4.

The DSCP code point settings for all subscription using the same transport session MUST be the same. Attempts to invoke `_establish-subscription_` with a different DSCP code point MUST be rejected.

If the publisher can satisfy the `_establish-subscription_` request, it replies with a numeric identifier for the subscription and then immediately starts streaming notification messages.

A dynamic subscription request MUST be declined if a publisher determines that it may be unable to provide update records meeting the terms of an `_establish-subscription_` RPC request.

Below is a tree diagram for `_establish-subscription_` YANG RPC. All objects contained in this tree are described in the YANG module in Section 12.2.

```

+---x establish-subscription {dynamic}?
|
|   +---w input
|   |
|   |   +---w name                subscription-name
|   |   +---w description?        string
|   |   +---w target
|   |   |
|   |   |   +---w datastore?      identityref
|   |   |   +---w (filter)
|   |   |   |   +---:(path)
|   |   |   |   |   +---w path      ypath
|   |   |   |   |   +---:(subtree)
|   |   |   |   |   |   +---w subtree  <anydata> {ypl:subtree}?
|   |   |   |   |   |   +---:(xpath)
|   |   |   |   |   |   |   +---w xpath  yang:xpath1.0 {ypl:xpath}?
|   |   |   +---w update-trigger
|   |   |   |   +---w periodic!
|   |   |   |   |   +---w period      centiseconds
|   |   |   |   |   +---w anchor-time? yang:date-and-time
|   |   |   |   +---w on-change! {on-change}?
|   |   |   |   |   +---w sync-on-start? boolean
|   |   |   +---w encoding          encoding
|   |   |   +---w dscp?             inet:dscp
|   |   +---ro output
|   |   |   +---ro id      subscription-id

```

Figure 14: `establish-subscription` YANG RPC

A publisher MAY reject the "establish-subscription" RPC for many reasons, as described in Section 2.4.6.

9.4.2. Deleting a Dynamic Subscription

The `_delete-subscription_` operation permits canceling an existing dynamic subscription that was established on the same transport session connecting to the subscriber.

The publisher responds to the request in the following way:

- * If the name identifier matches a `_dynamic_` subscription created on the same transport session then it MUST terminate the subscription, as per Section 9.1.4.

The publisher MAY reply back to the client before the subscription has been terminated, i.e., it may act asynchronously with respect to the delete request, however, the publisher MUST allow the

client to create a new subscription using the same name immediately after either the RPC operation completes or the `_subscription-terminated_` notification (Section 9.2.2) has been transmitted.

- * Otherwise, the request is failed with an "unknown subscription" error message.

Below is the tree diagram for the `_delete-subscription_` RPC. All objects contained in this tree are described in the YANG module in Section 12.2.

```
+---x delete-subscription {dynamic}?
|   +---w input
|       +---w name      subscription-name
```

Figure 15: delete-subscription YANG RPC

9.4.3. Killing a Dynamic Subscription

The `_kill-subscription_` RPC operation permits a client, that has the required access permissions, to forcibly terminate any arbitrary dynamic subscription, identified by subscription name, including those not associated with the transport session used for the `_kill-subscription_` RPC. The subscription is terminated as per Section 9.1.4.

The publisher MAY reply back to the client before the subscription has been terminated, i.e., it may act asynchronously with respect to the delete request, however, the publisher MUST allow the client to create a new subscription using the same name immediately after the `_subscription-terminated_` notification (Section 9.2.2) has been transmitted.

Below is the tree diagram for the `_kill-subscription_` RPC. All objects contained in this tree are described in the YANG module in Section 12.2.

```
+---x kill-subscription {dynamic}?
|   +---w input
|       +---w name      subscription-name
```

Figure 16: kill-subscription YANG RPC

9.4.4. RPC Failures

TODO, we should see if we can simplify how errors are reported.

Whenever an RPC is unsuccessful, the publisher returns relevant information as part of the RPC error response. Transport-level error processing **MUST** be done before the RPC error processing described in this section. In all cases, RPC error information returned by the publisher will use existing transport-layer RPC structures, such as those seen with NETCONF (Appendix A of [RFC6241]) or RESTCONF (Section 7.1 of [RFC8040]). These structures **MUST** be able to encode subscription-specific errors identified below and defined in this document's YANG data model.

As a result of this variety, how subscription errors are encoded in an RPC error response is transport dependent. Valid errors that can occur for each RPC are as follows:

establish-subscription	modify-subscription
-----	-----
dscp-unavailable	filter-unsupported
encoding-unsupported	insufficient-resources
filter-unsupported	no-such-subscription
insufficient-resources	
delete-subscription	kill-subscription
-----	-----
no-such-subscription	no-such-subscription

To see a NETCONF-based example of an error response from the list above, see the "no-such-subscription" error response illustrated in [RFC8640], Figure 10.

There is one final set of transport-independent RPC error elements included in the YANG data model defined in this document: three yang-data structures that enable the publisher to provide to the receiver any error information that does not fit into existing transport-layer RPC structures. These structures are:

1. "establish-subscription-stream-error-info": This **MUST** be returned with the leaf "reason" populated if an RPC error reason has not been placed elsewhere in the transport portion of a failed "establish-subscription" RPC response. This **MUST** be sent if hints on how to overcome the RPC error are included.
2. "modify-subscription-stream-error-info": This **MUST** be returned with the leaf "reason" populated if an RPC error reason has not been placed elsewhere in the transport portion of a failed "modify-subscription" RPC response. This **MUST** be sent if hints on how to overcome the RPC error are included.

3. "delete-subscription-error-info": This MUST be returned with the leaf "reason" populated if an RPC error reason has not been placed elsewhere in the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.

10. Robustness, Reliability, and Subscription Monitoring

10.1. Robustness and Reliability

It is important for clients to have confidence that the telemetry data that they receive is correct and complete. The design of YANG Push Lite achieves this in several ways:

- * the `_complete_` flag in `_update_` notification, or the equivalent `_update-complete_` notification, are used to signal when all data for a periodic collection event has been enqueued. This allows clients to delete stale information and monitor the performance and behavior of the publisher.
- * publishers use buffers when enqueueing traffic on to a transport to a receiver, but if that buffer becomes full then either the subscription is terminated, or in the case of a configured subscription with multiple receivers the slow receiver is disconnected from the subscription. In the case of dynamic subscriptions, the client may attempt to re-created the subscription. For configured subscriptions, the publisher should attempt to periodically recreate the subscription or reconnect the receiver.
 - *TODO, do we want to allow for a more lossy mode where the publisher just tail drops if the buffer is full? If we do, then this looks like it should be a configurable option.*
 - *TODO, ensure that we document how to reconnect to an new receiver*
- * the `_notification envelope_` structure, Section 7.1, used for all YANG Push notifications contains a monotonically increasing `_sequence-number_` field that allows for lost messages in an end-to-end data pipeline spanning multiple transport hops to be detected, allowing appropriate mitigation steps to be taken. For example, see figure 1 in [I-D.ietf-nmop-network-anomaly-architecture].
 - *TODO, sequence-numbers are unique except for the receiver-disconnected message.*

- * the protocol relies on transport protocols that are either reliable, e.g., TCP or HTTP, or unreliable transports that employ mechanisms for clients to detect when losses at the transport layer have occurred, e.g., the `_message id_` in [I-D.draft-ietf-netconf-udp-notif] (*TODO fix reference to bis version, once that becomes available*).
- * finally, if a publisher is not able to honor the expectation for a subscription for any reason, then the publisher always has the option to terminate the subscription. It can then subsequently refuse to handle new subscriptions if it does not have sufficient resources to handle the subscription.

10.2. Subscription Monitoring

In the operational state datastore, the `_datastore-telemetry_` container maintains operational state for all configured and dynamic subscriptions.

Dynamic subscriptions are only present in the `_datastore-telemetry/subscriptions/subscription_` list when they are active, and are removed as soon as they are terminated. Whereas configured subscriptions are present in the list if they are configured, regardless of whether they are active.

TODO, should dynamic receivers be listed? Do we need to report per-receiver stats for dynamic subscriptions?

The operational state is important for monitoring the health of subscriptions, receivers, and the overall telemetry subsystem.

This includes:

TODO, update the YANG model with more useful operational data, and mostly this section should briefly summarize and refer to the YANG model. We should also consider what indications to include from filters that cause a larger amount of internal work but don't generate a large number of transmitted notifications.

- * per subscription status and counters
- * per receiver status and counters
- * maybe some indication of the overall load on the telemetry subsystem, but we need to consider how useful that actually is, and whether just monitoring the device CPU load and general performance would be a better indication.

10.3. Publisher Capacity

It is far preferable to decline a subscription request than to accept such a request when it cannot be met.

Whether or not a subscription can be supported will be determined by a combination of several factors, such as the subscription update trigger (on-change or periodic), the period in which to report changes (one-second periods will consume more resources than one-hour periods), the amount of data in the datastore subtree that is being subscribed to, the number and combination of other subscriptions that are concurrently being serviced, and the overall load from other services running on the publisher.

11. Conformance and Capabilities

The normative text in this document already indicates which parts of the specification must or should be implemented for a compliant YANG Push Lite implementation via the use of [RFC2119] language. It also sets out some additional related requirements, e.g., on transports Section 8.2, that add in additional functionality.

Some parts of this specification are optional to implement. Some of these optional parts can be identified through the use of YANG Library [RFC8525] specifying the list of implemented YANG modules and YANG features. But, the broader approach adopted by this specification is via extending the ietf-system-capabilities YANG module specified in [RFC9196] to make capability information available as standard YANG described operational data.

11.1. Capabilities

Publishers SHOULD implement the ietf-system-capabilities YANG module, defined in [RFC9196], and the ietf-yp-lite-capabilities YANG module, defined in Section 14.2) that augments ietf-system-capabilities.

The ietf-yp-lite-capabilities module contains capabilities to indicate what types of subscriptions and transports may be configured, along with acceptable subscription parameter for given subtrees.

The schema tree for the ietf-system-capabilities augmented by ietf-yp-lite-capabilities is given below.

```

module: ietf-system-capabilities
  +--ro system-capabilities
    +--ro datastore-capabilities* [datastore]
      +--ro datastore
      |   -> /yanglib:yang-library/datastore/name
      +--ro per-node-capabilities* []
        +--ro (node-selection)?
        |   +--:(node-selector)
        |       +--ro node-selector?
        |           nacm:node-instance-identifier
        +--ro yplca:datastore-telemetry
          +--ro yplca:periodic-notifications-supported?
          |   notification-support
          +--ro (yplca:update-period)?
          |   +--:(yplca:minimum-update-period)
          |       |   +--ro yplca:minimum-update-period?           uint32
          |       +--:(yplca:supported-update-period)
          |           +--ro yplca:supported-update-period*         uint32
          +--ro yplca:on-change-supported?
          |   notification-support
        +--ro yplca:datastore-telemetry
          +--ro yplca:periodic-notifications-supported?
          |   notification-support
          +--ro (yplca:update-period)?
          |   +--:(yplca:minimum-update-period)
          |       |   +--ro yplca:minimum-update-period?           uint32
          |       +--:(yplca:supported-update-period)
          |           +--ro yplca:supported-update-period*         uint32
          +--ro yplca:on-change-supported?
          |   notification-support
        +--ro yplca:transport
          +--ro yplca:transport-capability* [transport-protocol]
          |   +--ro yplca:transport-protocol           identityref
          |   +--ro yplca:security-protocol?           identityref
          |   +--ro yplca:encoding-format*             identityref

```

Figure 17: YANG tree for ietf-system-capabilities with ietf-yl-lite-capabilities augmentations.

*TODO, do we need additional capabilities, as per Issue 18
 (<https://github.com/rgwilton/draft-yp-observability/issues/18>)*

11.2. Subscription Content Schema Identification

YANG Module Synchronization

To make subscription requests, the subscriber needs to know the YANG datastore schemas used by the publisher. These schemas are available in the YANG library module `ietf-yang-library.yang` as defined in [RFC8525]. The receiver is expected to know the YANG library information before starting a subscription.

The set of modules, revisions, features, and deviations can change at runtime (if supported by the publisher implementation). For this purpose, the YANG library provides a simple "yang-library-change" notification that informs the subscriber that the library has changed. In this case, a subscription may need to be updated to take the updates into account. The receiver may also need to be informed of module changes in order to process updates regarding datastore

TODO, this section should be updated so that a subscription is restarted if the schema that it is using changes, and to incorporate ideas to fingerprint the subscription schema in the subscription-started notification.

12. Core YANG Push Lite YANG Data Model

12.1. ietf-yp-lite YANG tree

This section shows the full tree output for `ietf-yp-lite` YANG module.

Note, this output does not include support for any transport configuration, and for any implementation that supports configured subscriptions using this YANG module then at least one transport would expect to be configurable.

module: `ietf-yp-lite`

```

rpcs:
  +---x establish-subscription {dynamic}?
  |   +---w input
  |   |   +---w name          subscription-name
  |   |   +---w description?   string
  |   |   +---w target
  |   |   |   +---w datastore?   identityref
  |   |   |   +---w (filter)
  |   |   |   |   +--:(path)
  |   |   |   |   |   +---w path      ypath
  |   |   |   |   |   +--:(subtree)
  |   |   |   |   |   |   +---w subtree  <anydata> {ypl:subtree}?
  |   |   |   |   |   |   +--:(xpath)
  |   |   |   |   |   |   |   +---w xpath  yang:xpath1.0 {ypl:xpath}?
  |   |   +---w update-trigger
  |   |   +---w periodic!

```

```

| | | | +---w period                centiseconds
| | | | +---w anchor-time?         yang:date-and-time
| | | | +---w on-change! {on-change}?
| | | | +---w sync-on-start?       boolean
| | | +---w encoding                encoding
| | | +---w dscp?                   inet:dscp
| | +--ro output
| | +--ro id                        subscription-id
+---x delete-subscription {dynamic}?
| +---w input
| | +---w name                      subscription-name
+---x kill-subscription {dynamic}?
| +---w input
| | +---w name                      subscription-name

notifications:
+---n subscription-started
| +--ro id                          subscription-id
| +--ro name                        subscription-name
| +--ro description?                string
| +--ro target
| | +--ro datastore?                identityref
| | +--ro (filter)
| | | +---:(path)
| | | | +--ro path                  ypath
| | | +---:(subtree)
| | | | +--ro subtree               <anydata> {ypl:subtree}?
| | | +---:(xpath)
| | | | +--ro xpath                 yang:xpath1.0 {ypl:xpath}?
+--ro update-trigger
| +--ro periodic!
| | +--ro period                    centiseconds
| | +--ro anchor-time?              yang:date-and-time
| +--ro on-change! {on-change}?
| +--ro sync-on-start?              boolean
+---n subscription-terminated
| +--ro name                        subscription-name
| +--ro id                          subscription-id
| +--ro reason                       identityref
+---n receiver-disconnected
| +--ro name                        subscription-name
| +--ro id                          subscription-id
| +--ro reason                       identityref
+---n update
| +--ro id?                         subscription-id
| +--ro path-prefix?                 string
| +--ro snapshot-type?               enumeration
| +--ro observation-time?             yang:date-and-time

```

```

|   |--ro updates* [target-path]
|   |   |--ro target-path    string
|   |   |--ro data?         <anydata>
|   |--ro complete?         empty
+---n update-complete
    |--ro id      subscription-id

```

Figure 18: YANG tree for YANG Push Lite Module Tree Output

12.2. ietf-yp-lite YANG Model

This module imports typedefs from [RFC6991], [RFC8343], [RFC8341], [RFC8529], and [RFC8342]. It references [RFC6241], [XPATH] ("XML Path Language (XPath) Version 1.0"), [RFC7049], [RFC8259], [RFC7950], [RFC7951], and [RFC7540].

This YANG module imports typedefs from [RFC6991], identities from [RFC8342], and the "sx:structure" extension from [RFC8791]. It also references [RFC6241], [XPATH], and [RFC7950].

```

<CODE BEGINS> file "ietf-yp-lite.yang#0.1.0"
module ietf-yp-lite {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yp-lite";
  prefix ypl;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8525: YANG Data Structure Extensions";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-datastores {
    prefix ds;

```

```
reference
  "RFC 8342: Network Management Datastore Architecture (NMDA)";
}

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

  Author:   Robert Wilton
            <mailto:rwilton@cisco.com>";
description
  "This module contains YANG specifications for YANG-Push lite,
  a simplified version of the YANG-Push [RFC 8641] protocol.

  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) 2025 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 2025-08-03 {
  description
    "Initial revision.";
  reference
    "XXXX: YANG Push Lite";
}

/*
 * FEATURES
 */

feature dynamic {
```

```
    description
      "This feature indicates that dynamic establishment of
      subscriptions is
      supported.";
  }

  feature on-change {
    description
      "This feature indicates that on-change triggered subscriptions
      are supported.";
  }

  feature subtree {
    description
      "This feature indicates support for YANG subtree filtering.";
    reference
      "RFC 6241: Network Configuration Protocol (NETCONF),
      Section 6";
  }

  feature xpath {
    description
      "This feature indicates support for XPath filtering.";
    reference
      "XML Path Language (XPath) Version 1.0
      (https://www.w3.org/TR/1999/REC-xpath-19991116)";
  }

/*
 * IDENTITIES
 */
/* Identities for RPC and notification errors */

  identity delete-subscription-error {
    description
      "Base identity for the problem found while attempting to
      fulfill either a 'delete-subscription' RPC request or a
      'kill-subscription' RPC request.";
  }

  identity establish-subscription-error {
    description
      "Base identity for the problem found while attempting to
      fulfill an 'establish-subscription' RPC request.";
  }

  identity subscription-terminated-reason {
    description
```

```
    "Base identity for the problem condition communicated to a
    receiver as part of a 'subscription-terminated'
    notification.";
}

identity dscp-unavailable {
    base establish-subscription-error;
    description
        "The publisher is unable to mark notification messages with
        prioritization information in a way that will be respected
        during network transit.";
}

identity encoding-unsupported {
    base establish-subscription-error;
    description
        "Unable to encode notification messages in the desired
        format.";
}

identity filter-unavailable {
    base subscription-terminated-reason;
    description
        "Referenced filter does not exist. This means a receiver is
        referencing a filter that doesn't exist or to which it
        does not have access permissions.";
}

identity filter-unsupported {
    base establish-subscription-error;
    description
        "Cannot parse syntax in the filter. This failure can be from
        a syntax error or a syntax too complex to be processed by the
        publisher.";
}

identity insufficient-resources {
    base establish-subscription-error;
    description
        "The publisher does not have sufficient resources to support
        the requested subscription. An example might be that
        allocated CPU is too limited to generate the desired set of
        notification messages.";
}

identity no-such-subscription {
    base delete-subscription-error;
    base subscription-terminated-reason;
```

```
    description
      "Referenced subscription doesn't exist. This may be as a
       result of a nonexistent subscription ID, an ID that belongs to
       another subscriber, or an ID for a configured subscription.";
  }

  identity stream-unavailable {
    base subscription-terminated-reason;
    description
      "Not a subscribable event stream. This means the referenced
       event stream is not available for subscription by the
       receiver.";
  }

  identity suspension-timeout {
    base subscription-terminated-reason;
    description
      "Termination of a previously suspended subscription. The
       publisher has eliminated the subscription, as it exceeded a
       time limit for suspension.";
  }

  identity unsupportable-volume {
    base subscription-terminated-reason;
    description
      "The publisher does not have the network bandwidth needed to
       get the volume of generated information intended for a
       receiver.";
  }

/* Identities for encodings */

  identity configurable-encoding {
    description
      "If a transport identity derives from this identity, it means
       that it supports configurable encodings. An example of a
       configurable encoding might be a new identity such as
       'encode-cbor'. Such an identity could use
       'configurable-encoding' as its base. This would allow a
       dynamic subscription encoded in JSON (RFC 8259) to request
       that notification messages be encoded via the Concise Binary
       Object Representation (CBOR) (RFC 7049). Further details for
       any specific configurable encoding would be explored in a
       transport document based on this specification.

       TODO - Clear up this text or use YANG-CBOR reference";
    reference
      "RFC 8259: The JavaScript Object Notation (JSON) Data
```

```
        Interchange Format
        RFC 7049: Concise Binary Object Representation (CBOR)";
    }

    identity encoding {
        description
            "Base identity to represent data encodings.";
    }

    identity json {
        base encoding;
        description
            "Encode data using JSON as described in RFC 7951.";
        reference
            "RFC 7951: JSON Encoding of Data Modeled with YANG";
    }

    identity xml {
        base encoding;
        description
            "Encode data using XML as described in RFC 7950.";
        reference
            "RFC 7950: The YANG 1.1 Data Modeling Language";
    }

    identity cbor {
        base encoding;
        description
            "Encode data using CBOR as described in RFC 9245,
            without using YANG SIDs";
        reference
            "RFC 9245: Encoding of Data Modeled with YANG in the
            Concise Binary Object Representation (CBOR).";
    }

    identity cbor-sids {
        base encoding;
        description
            "Encode data using JSON as described in RFC 7951, using YANG
            SIDs.";
        reference
            "RFC 9245: Encoding of Data Modeled with YANG in the
            Concise Binary Object Representation (CBOR).

            RFC 9595: YANG Schema Item iDentifier (YANG SID).";
    }

    /* Identities for transports */
```



```
identity transport {
  description
    "An identity that represents the underlying mechanism for
    passing notification messages.";
}

/*
 * TYPEDEFS
 */

typedef ypath {
  type string {
    length "1..max";
  }
  description
    "A type for YANG instance data paths.";
}

typedef encoding {
  type identityref {
    base encoding;
  }
  description
    "Specifies a data encoding, e.g., for a data subscription.";
}

typedef subscription-name {
  type string {
    length "1..64";
    // TODO, is it reasonable to limit this to 64 characters?
  }
  description
    "A user friendly name for a subscription.";
}

typedef subscription-id {
  type uint32;
  description
    "A type for subscription identifiers.";
}

typedef transport {
  type identityref {
    base transport;
  }
  description
    "Specifies the transport used to send notification messages
```

```
        to a receiver.";
    }

// TODO - Consider changes to list keys or reordering of
//         user-ordered lists.

typedef centiseconds {
    type uint32;
    description
        "A period of time, measured in units of 0.01 seconds.";
}

typedef subscription-type {
    type enumeration {
        enum configured {
            description
                "A subscription that is created and managed via
                configuration.";
        }
        enum dynamic {
            description
                "A subscription that is created and managed via RPC
                primitives.";
        }
    }
    description
        "Indicate the type of subscription.";
}

typedef subscription-status {
    type enumeration {
        enum invalid {
            description
                "The subscription as a whole is unsupportable with its
                current parameters.";
        }
        enum inactive {
            description
                "The subscription is supportable with its current
                parameters, but it is not currently connected to any
                connected receivers";
        }
        enum active {
            description
                "The subscription is actively running, and is connected
                to at least one receiver.

                A subscription-started notification must have been sent
```

```
        for a subscription to be in this state, and the receiver
        will receive update notifications, as per the
        update-trigger selection.";
    }
}
description
    "Indicates the status of a subscription";
}

/*
 * GROUPINGS
 */

grouping dscp {
    description
        "This grouping describes QoS information concerning a
        subscription. This information is passed to lower layers
        for transport prioritization and treatment.";
    leaf dscp {
        type inet:dscp;
        default "0";
        description
            "The desired network transport priority level. This is the
            priority set on notification messages encapsulating the
            results of the subscription. This transport priority is
            shared for all receivers of a given subscription.";
    }
}

grouping filter-choice {
    description
        "This grouping defines the types of selectors for objects
        from a datastore.";
    choice filter {
        mandatory true;
        description
            "The content filter specification for this request.";

        leaf path {
            type ypath;
            mandatory true;
            description
                "A basic path filter that allows wildcard, regex, or
                fixed value for list keys. Each format is TODO";
        }
    }
    anydata subtree {
        if-feature "ypl:subtree";
        mandatory true;
    }
}
```

```
description
  "This parameter identifies the portions of the
  target datastore to retrieve.";
reference
  "RFC 6241: Network Configuration Protocol (NETCONF),
  Section 6";
}
leaf xpath {
  if-feature "ypl:xpath";
  type yang:xpath1.0;
  mandatory true;
  description
    "This parameter contains an XPath expression identifying
    the portions of the target datastore to retrieve.

    If the expression returns a node set, all nodes in the
    node set are selected by the filter. Otherwise, if the
    expression does not return a node set, the filter
    doesn't select any nodes.

    The expression is evaluated in the following XPath
    context:

    o The set of namespace declarations is the set of prefix
      and namespace pairs for all YANG modules implemented
      by the server, where the prefix is the YANG module
      name and the namespace is as defined by the
      'namespace' statement in the YANG module.

      If the leaf is encoded in XML, all namespace
      declarations in scope on the 'stream-xpath-filter'
      leaf element are added to the set of namespace
      declarations. If a prefix found in the XML is
      already present in the set of namespace declarations,
      the namespace in the XML is used.

    o The set of variable bindings is empty.

    o The function library is comprised of the core
      function library and the XPath functions defined in
      Section 10 in RFC 7950.

    o The context node is the root node of the target
      datastore.";
```

```
reference
  "XML Path Language (XPath) Version 1.0
  (https://www.w3.org/TR/1999/REC-xpath-19991116)
  RFC 7950: The YANG 1.1 Data Modeling Language,
```

```

        Section 10";
    }
}

grouping update-policy {
  description
    "This grouping describes the susbcription update policy";

  container update-trigger {
    description
      "This container describes all conditions under which
       subscription update messages are generated";

    container periodic {
      presence "indicates a periodic subscription";
      description
        "The publisher is requested to periodically notify the
         receiver regarding the current values of the datastore
         as defined by the selection filter.";
      leaf period {
        type centiseconds;
        mandatory true;
        description
          "Duration of time that should occur between periodic
           push updates, in units of 0.01 seconds.";
      }
      leaf anchor-time {
        type yang:date-and-time;
        description
          "Designates a timestamp before or after which a series
           of periodic push updates are determined. The next
           update will take place at a point in time that is a
           multiple of a period from the 'anchor-time'.
           For example, for an 'anchor-time' that is set for the
           top of a particular minute and a period interval of a
           minute, updates will be sent at the top of every
           minute that this subscription is active.";
      }
    }
  }
  container on-change {
    if-feature "on-change";
    presence "indicates an on-change subscription";
    description
      "The publisher is requested to notify the receiver
       regarding changes in values in the datastore subset as
       defined by a selection filter.";
  }
}
```

```
leaf sync-on-start {
  type boolean;
  default "true";
  description
    "When this object is set to 'false', (1) it restricts an
    on-change subscription from sending 'push-update'
    notifications and (2) pushing a full selection per the
    terms of the selection filter MUST NOT be done for
    this subscription. Only updates about changes
    (i.e., only 'push-change-update' notifications)
    are sent. When set to 'true' (the default behavior),
    in order to facilitate a receiver's synchronization,
    a full update is sent, via a 'push-update' notification,
    when the subscription starts. After that,
    'push-change-update' notifications are exclusively sent,
    unless the publisher chooses to resync the subscription
    via a new 'push-update' notification.";
}
}
}

grouping subscription-common {
  description
    "Common settings that are shared between dynamic and
    configured subscriptions.";

  leaf name {
    type subscription-name;
    mandatory true;
    description
      "The client provided name for the subscription.

      This MUST be unique across all subscriptions. Configuring
      a subscription with a name already used by a dynamic
      subscription will replace the dynamic subscription, forcing
      it to be terminated.";
  }

  leaf description {
    type string {
      length "1..1000";
    }
    description
      "Open text allowing a configuring entity to embed the
      originator or other specifics of this subscription.";
  }
}
```

```
    container target {
      description
        "Identifies the source of information against which a
        subscription is being applied as well as specifics on the
        subset of information desired from that source.";
      leaf datastore {
        type identityref {
          base ds:datastore;
        }
        default "ds:operational";
        description
          "Datastore from which to retrieve data, defaults to
          operational";
      }

      uses filter-choice;
    }

    uses update-policy;
  }

/*
 * RPCs
 */

rpc establish-subscription {
  if-feature "dynamic";
  description
    "This RPC allows a subscriber to create (and possibly
    negotiate) a subscription on its own behalf.  If successful,
    the subscription remains in effect for the duration of the
    subscriber's association with the publisher or until the
    subscription is terminated.  If an error occurs or the
    publisher cannot meet the terms of a subscription, an RPC
    error is returned, and the subscription is not created.
    In that case, the RPC reply's 'error-info' MAY include
    suggested parameter settings that would have a higher
    likelihood of succeeding in a subsequent
    'establish-subscription' request.";
  input {
    uses subscription-common;

    leaf encoding {
      type encoding;
      mandatory true;
      description
        "The encoding to use for the subscription notifications.";
    }
  }
}
```

```
    }

    uses dscp;

}

output {
  leaf id {
    type subscription-id;
    mandatory true;
    description
      "Identifier used for this subscription.";
  }
}

}

sx:structure establish-subscription-stream-error-info {
  container establish-subscription-stream-error-info {
    description
      "If any 'establish-subscription' RPC parameters are
      unsupportable against the event stream, a subscription
      is not created and the RPC error response MUST indicate the
      reason why the subscription failed to be created. This
      yang-data MAY be inserted as structured data in a
      subscription's RPC error response to indicate the reason for
      the failure. This yang-data MUST be inserted if hints are
      to be provided back to the subscriber.";
    leaf reason {
      type identityref {
        base establish-subscription-error;
      }
      description
        "Indicates the reason why the subscription has failed to
        be created to a targeted event stream.";
    }
    leaf filter-failure-hint {
      type string;
      description
        "Information describing where and/or why a provided
        filter was unsupportable for a subscription. The
        syntax and semantics of this hint are
        implementation specific.";
    }
  }
}

}

rpc delete-subscription {
  if-feature "dynamic";
  description
```


"This RPC allows a subscriber to delete a subscription that was previously created by that same subscriber using the 'establish-subscription' RPC.

Only subscriptions that were created using 'establish-subscription' from the same origin as this RPC can be deleted via this RPC. // TODO - Why same origin?

If an error occurs, the server replies with an 'rpc-error' where the 'error-info' field MAY contain a 'delete-subscription-error-info' structure.";

```
input {
  leaf name {
    type subscription-name;
    mandatory true;
    description
      "The name of the dynamic subscription to be deleted.";
  }
}

rpc kill-subscription {
  if-feature "dynamic";
  nacm:default-deny-all;
  description
    "This RPC allows an operator to delete a dynamic subscription
    without restrictions on the originating subscriber or
    underlying transport session.

    Only dynamic subscriptions, i.e., those that were created
    using 'establish-subscription', may be deleted via this RPC.

    If an error occurs, the server replies with an 'rpc-error'
    where the 'error-info' field MAY contain a
    'delete-subscription-error-info' structure.";
  input {
    leaf name {
      type subscription-name;
      mandatory true;
      description
        "The name of the dynamic subscription to be deleted.";
    }
  }
}

sx:structure delete-subscription-error-info {
  container delete-subscription-error-info {
    description
```

```
    "If a 'delete-subscription' RPC or a 'kill-subscription' RPC
    fails, the subscription is not deleted and the RPC error
    response MUST indicate the reason for this failure.  This
    yang-data MAY be inserted as structured data in a
    subscription's RPC error response to indicate the reason
    for the failure.";
  leaf reason {
    type identityref {
      base delete-subscription-error;
    }
    mandatory true;
    description
      "Indicates the reason why the subscription has failed to be
      deleted.";
  }
}

/*
 * NOTIFICATIONS
 */

notification subscription-started {
  //ypl:subscription-state-notification;
  description
    "This notification indicates that a configured subscription
    has started and notifications will now be sent.";
  leaf id {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
  uses subscription-common;
}

notification subscription-terminated {
  //ypl:subscription-state-notification;
  description
    "This notification indicates that a subscription has been
    terminated.";
  leaf name {
    type subscription-name;
    mandatory true;
    description
      "The name of the subscription that has been terminated.";
  }
  leaf id {
```

```
    type subscription-id;
    mandatory true;
    description
        "This references the affected subscription.";
}
leaf reason {
    type identityref {
        base subscription-terminated-reason;
    }
    mandatory true;
    description
        "Identifies the condition that resulted in the
        termination.";
}
}

notification receiver-disconnected {
    description
        "This notification indicates that a receiver has been
        disconnected from a subscription. This notification is only
        sent to the affected receiver if the transport is still
        available, and only if the subscription remains active due
        to other active receivers.

        The receiver will not receive any further notifications
        for this subscription unless it receives a new
        'subscription-started' notification for this
        subscription.";
    leaf name {
        type subscription-name;
        mandatory true;
        description
            "The name of the subscription.";
    }
    leaf id {
        type subscription-id;
        mandatory true;
        description
            "The subscription identifier";
    }
    leaf reason {
        type identityref {
            base subscription-terminated-reason;
            // TODO, add new receiver disconnected reasons.
        }
        mandatory true;
        description
            "Identifies the condition that resulted in the
```

```
        termination.";
    }
}

notification update {
  description
    "This notification contains a push update that in turn
    contains data subscribed to via a subscription. In the case
    of a periodic subscription, this notification is sent for
    periodic updates. It can also be used for synchronization
    updates of an on-change subscription. This notification
    shall only be sent to receivers of a subscription.";
  leaf id {
    type subscription-id;
    description
      "This references the subscription that drove the
      notification to be sent.";
  }

  leaf path-prefix {
    type string;
    description
      "Specifies the common prefix that all other paths and data
      are encoded relative to.

      TODO - This should be a JSONified instance data path.";
  }

  leaf snapshot-type {
    type enumeration {
      enum "periodic" {
        description
          "The update message is due to a periodic update.";
      }
      enum "on-change-update" {
        description
          "The update message is due to an on-change update. This
          means that one or more fields have changed under the
          snapshot path.

          TODO - Split this into a on-change-delete msg?";
      }
      enum "on-change-delete" {
        description
          "The update message is due to an on-change event where
          the data node at the target path has been delete.";
      }
      enum "resync" {
```

```
    description
      "This indicates that the update is to resynchronize the
       state, e.g., after a subscription started notification.

       Ideally, the resync message SHOULD be the first
       notification sent when a subscription has started, but
       it is not gauranteed or required to be the first
       (e.g., if an on-change event occurs).

       These messages can be used to ensure that all state
       has been sent to the client, and can be used to purge
       stale data.

       TODO - In the distributed notification case, need a
       notification to indicate that all child subscriptions
       have been sent.";
  }
}
description
  "This indicates the type of notification message that is
   being sent.";
}

leaf observation-time {
  type yang:date-and-time;
  description
    "The time that the update was observed by the publisher.";
}

list updates {
  key "target-path";
  description
    "This list contains the updated data. It constitutes a
     snapshot at the time of update of the set of data that has
     been subscribed to. The snapshot corresponds to the same
     snapshot that would be returned in a corresponding 'get'
     operation with the same selection filter parameters
     applied.";

  leaf target-path {
    type string;
    description
      "The target path of the data that is being replaced, i.e.,
       updated or deleted. The path is given relative to the
       path-prefix.";
  }
}

anydata data {
```

```
description
  "This contains the updated data. It constitutes a
  snapshot at the time of update of the set of data that has
  been subscribed to. The snapshot corresponds to the same
  snapshot that would be returned in a corresponding 'get'
  operation with the same selection filter parameters
  applied.

  For an on-change delete notification, the
  datastore-snapshot will be absent for the given target
  path.

  The snapshot is encoded relative to the path-prefix.";
}
}

leaf complete {
  type empty;
  description
    "This flag indicates that this is the last
    notification in a series of notifications that together
    constitute a complete snapshot of the subscribed data at
    the event-time.

    The 'update-complete' notification MAY be used as an
    alternative to setting this flag, and is semantically
    equivalent.";
}

// TODO - Need to signal when an initial replay has completed, or
//          possibly when any period subscription has completed.
// TODO - Need to think about list key entries that are no longer
//          present.
notification update-complete {
  //ypl:subscription-state-notification;
  //if-feature "replay";
  description
    "This notification indicates the end of a periodic collection
    or resync event for an on-change subscription, and can be used
    to indicate that the receiver has a complete snapshot of the
    subscribed data at the event-time, and hence the client MAY
    use this as a signal that it can purge stale state
    information.

    Alternatively, the 'complete' flag in the update
    notification MAY be used instead of sending this notification
    as a separate message, and both are semantically equivalent.";
```

```

    leaf id {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }
}
<CODE ENDS>

```

Figure 19: YANG module ietf-yp-lite

13. Configured Subscription YANG Data Model

This document specifies the ietf-yp-lite-config YANG module Section 13.2 that defines an NMDA [RFC8342] compatible YANG data model for configuring subscriptions. Support for this YANG module is OPTIONAL and is advertised using the normal mechanisms, e.g., [RFC8525].

Below is a tree diagram for the "subscriptions" container. All objects contained in this tree are described in the YANG module in Section 12.2. In the operational datastore [RFC8342], the "subscription" list contains entries both for configured and dynamic subscriptions.

```

module: ietf-yp-lite-config
  +--rw datastore-telemetry!
    +--rw subscriptions
      +--rw subscription* [name]
        +--rw name
          | subscription-name
        +--rw description? string
        +--rw target
          | +--rw datastore? identityref
          | +--rw (filter)
          | | +--:(path)
          | | | +--rw path ypath
          | | +--:(subtree)
          | | | +--rw subtree <anydata> {ypl:subtree}?
          | | +--:(xpath)
          | | | +--rw xpath yang:xpath1.0 {ypl:xpath}?
          | | +--:(by-reference)
          | | | +--rw filter-ref filter-ref
        +--rw update-trigger
          | +--rw periodic!
          | | +--rw period centiseconds
          | | +--rw anchor-time? yang:date-and-time

```

```

    |   +--rw on-change! {on-change}?
    |   +--rw sync-on-start?   boolean
+--rw receivers* [name]
    |   +--rw name
    |   |       -> /datastore-telemetry/receivers/receiver/name
    |   +--ro status?          enumeration
    |   +--ro statistics
    |   |       +--ro receiver-disconnected-count?
    |   |       |       yang:zero-based-counter64
+--ro id
    |       ypl:subscription-id
+--ro status?
    |       ypl:subscription-status
+--ro last-sequence-number?
    |       yang:zero-based-counter64
+--ro last-notification-time?
    |       yang:date-and-time
+--ro last-periodic-collection-time?
    |       yang:date-and-time
+--ro last-on-change-notification-time?
    |       yang:date-and-time
+--ro statistics
    |   +--ro started-notification-count?
    |   |       yang:zero-based-counter64
    |   +--ro terminated-notification-count?
    |   |       yang:zero-based-counter64
    |   +--ro update-notification-count?
    |   |       yang:zero-based-counter64
    |   +--ro periodic-collection-count?
    |   |       yang:zero-based-counter64
    |   +--ro excluded-event-records?
    |   |       yang:zero-based-counter64
+---x reset

augment /ypl:establish-subscription/ypl:input/ypl:target
    /ypl:filter:
    +--:(by-reference)
augment /ypl:subscription-started/ypl:target:
    +-- filter-ref?   filter-ref

```

Figure 20: subscriptions container Tree Diagram

An overview of the behavior for configured subscriptions is specified in Section 9.3, with further details specified in the ietf-yp-lite-config YANG module.

13.1. ietf-yp-lite-config YANG tree

This section shows the full tree output for ietf-yp-lite-config YANG module.

Note, this output does not include support for any transport configuration, and for any implementation that supports configured subscriptions using this YANG module then at least one transport would expect to be configurable.

```

module: ietf-yp-lite-config
  +--rw datastore-telemetry!
    +--rw filters
      +--rw filter* [name]
        +--rw name string
        +--rw (filter)
          +--:(path)
            | +--rw path ypath
          +--:(subtree)
            | +--rw subtree <anydata> {ypl:subtree}?
          +--:(xpath)
            | +--rw xpath yang:xpath1.0 {ypl:xpath}?
    +--rw subscriptions
      +--rw subscription* [name]
        +--rw name
          | subscription-name
        +--rw description? string
        +--rw target
          | +--rw datastore? identityref
          +--rw (filter)
            +--:(path)
              | +--rw path ypath
            +--:(subtree)
              | +--rw subtree <anydata> {ypl:subtree}?
            +--:(xpath)
              | +--rw xpath yang:xpath1.0 {ypl:xpath}?
            +--:(by-reference)
              | +--rw filter-ref filter-ref
        +--rw update-trigger
          | +--rw periodic!
          | | +--rw period centiseconds
          | | +--rw anchor-time? yang:date-and-time
          +--rw on-change! {on-change}?
            | +--rw sync-on-start? boolean
        +--rw receivers* [name]
          | +--rw name
          | | -> /datastore-telemetry/receivers/receiver/name
          +--ro status? enumeration

```

```

|         |--ro statistics
|         |   |--ro receiver-disconnected-count?
|         |       yang:zero-based-counter64
|--ro id
|   ypl:subscription-id
|--ro status?
|   ypl:subscription-status
|--ro last-sequence-number?
|   yang:zero-based-counter64
|--ro last-notification-time?
|   yang:date-and-time
|--ro last-periodic-collection-time?
|   yang:date-and-time
|--ro last-on-change-notification-time?
|   yang:date-and-time
|--ro statistics
|   |--ro started-notification-count?
|   |   yang:zero-based-counter64
|   |--ro terminated-notification-count?
|   |   yang:zero-based-counter64
|   |--ro update-notification-count?
|   |   yang:zero-based-counter64
|   |--ro periodic-collection-count?
|   |   yang:zero-based-counter64
|   |--ro excluded-event-records?
|   |   yang:zero-based-counter64
|   +---x reset
+---rw receivers
  +---rw receiver* [name]
    +---rw name                               string
    +---rw encoding                           ypl:encoding
    +---rw dscp?                              inet:dscp
    +---x reset
    +---rw (notification-message-origin)?
      +---:(interface-originated)
      |   +---rw source-interface?   if:interface-ref
      |       {interface-designation}?
      +---:(address-originated)
      |   +---rw source-vrf?         leafref {supports-vrf}?
      |   +---rw source-address?     inet:ip-address-no-zone
    +---rw (transport-type)

augment /ypl:establish-subscription/ypl:input/ypl:target
  /ypl:filter:
    +---:(by-reference)
    +--- filter-ref?   filter-ref
augment /ypl:subscription-started/ypl:target:
  +--- filter-ref?   filter-ref

```

Figure 21: YANG tree for YANG Push Lite Config Module Tree Output

13.2. ietf-yp-lite-config YANG Model

This module has import dependencies on [RFC6991], [RFC8343], and [RFC8529], and ietf-yang-push-lite.yang (this RFC). In addition, this YANG module references [BCP14] ([RFC2119] [RFC8174]), and [RFC8529].

```
<CODE BEGINS> file "ietf-yp-lite-config.yang#0.1.0"
module ietf-yp-lite-config {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yp-lite-config";
  prefix yplco;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }
  import ietf-network-instance {
    prefix ni;
    reference
      "RFC 8529: YANG Data Model for Network Instances";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-yp-lite {
    prefix ypl;
    reference
      "RFC XXXX: YANG Push Lite";
  }

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

    Author:   Robert Wilton
```

```
<mailto:rwilton@cisco.com>";
description
  "This module contains YANG specifications for YANG-Push lite
  configuration and operational state model.

  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) 2025 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 2025-08-03 {
  description
    "Initial revision.";
  reference
    "XXX: YANG Push Lite";
}

/*
 * FEATURES
 */

feature interface-designation {
  description
    "This feature indicates that a publisher supports sourcing all
    receiver interactions for a configured subscription from a
    single designated egress interface.";
}

feature supports-vrf {
  description
    "This feature indicates that a publisher supports VRF
    configuration for configured subscriptions. VRF support for
    dynamic subscriptions does not require this feature.";
```

```
reference
  "RFC 8529: YANG Data Model for Network Instances,
    Section 6";
}

/*
 * TYPEDEFS
 */

typedef filter-ref {
  type leafref {
    path "/datastore-telemetry/filters/filter/name";
  }
  description
    "This type is used to reference a selection filter.";
}

/*
 * GROUPINGS
 */

grouping filter-ref {
  description
    "This grouping is used to reference a selection filter.";
  leaf filter-ref {
    type filter-ref;
    mandatory true;
    description
      "References an existing selection filter that is to be
        applied to the subscription.";
  }
}

/*
 * DATA NODES
 */

container datastore-telemetry {
  presence "Enables datastore telemetry";
  description
    "YANG Push Lite Datastore Telemetry Configuration and State.";
  container filters {
    description
      "Contains a list of configurable filters that can be applied
        to subscriptions. This facilitates the reuse of complex
        filters once defined.";

    list filter {
      key "name";
    }
  }
}
```

```
    description
      "A list of preconfigured filters that can be applied
      to datastore subscriptions.";
    leaf name {
      type string;
      description
        "A unique name to identify the selection filters.";
    }
    uses ypl:filter-choice;
  }
}
container subscriptions {
  description
    "Contains the list of currently active subscriptions, i.e.,
    subscriptions that are currently in effect, used for
    subscription management and monitoring purposes. This
    includes subscriptions that have been set up via
    RPC primitives as well as subscriptions that have been
    established via configuration.";
  list subscription {
    key "name";
    description
      "The identity and specific parameters of a subscription.
      Subscriptions in this list can be created using a control
      channel or RPC or can be established through configuration.

      If the 'kill-subscription' RPC or configuration operations
      are used to delete a subscription, a
      'subscription-terminated' message is sent to any active or
      suspended receivers.";

    uses ypl:subscription-common;

    list receivers {
      key "name";
      min-elements 1;
      description
        "A host intended as a recipient for the notification
        messages of a subscription. For configured
        subscriptions, transport-specific network parameters
        (or a leafref to those parameters) may be augmented to a
        specific receiver in this list.";
      leaf name {
        type leafref {
          path "/datastore-telemetry/receivers/receiver/name";
        }
        description
          "Identifies a unique receiver for a subscription.";
      }
    }
  }
}
```

```
}  
  
leaf status {  
  type enumeration {  
    enum disconnected {  
      description  
        "This subscription does not have an active session  
        with the receiver, and it is not trying to  
        connect.  
  
        E.g., this state may be reported if the  
        subscription is not valid, or has not been started  
        yet.";  
    }  
    enum connecting {  
      description  
        "The publisher is trying to establish a session  
        with the receiver for this subscription.  
  
        For a session less transport, this state may be  
        used to indicate that there is no route to the  
        receiver.  
  
        A receiver in connecting state may indicate that  
        the transport or associated security session  
        could not be established, and the publisher is  
        periodically trying to establish the connection.";  
    }  
    enum active {  
      description  
        "The publisher has successfully connected (if over  
        a session based transport) to the receiver for  
        this subscription, and the publisher is able to  
        send notifications to the receiver.";  
    }  
  }  
  config false;  
  description  
    "Specifies the connection status of the receiver for  
    this subscription.";  
}  
  
container statistics {  
  config false;  
  description  
    "Statistics related to the number of messages sent to  
    this specific receiver for this subscription.";
```

```
    leaf receiver-disconnected-count {
      type yang:zero-based-counter64;
      config false;
      description
        "The number of times that the receiver has been
        disconnected since the subscription
        receiver entry was created and the configuration
        was applied by the publisher";
    }
  }
}

leaf id {
  type ypl:subscription-id;
  config false;
  mandatory true;
  description
    "Publisher allocated identifier for a subscription;
    Unique in a given publisher.";
}

leaf status {
  type ypl:subscription-status;
  config false;
  description
    "The presence of this leaf indicates that the
    subscription originated from configuration, not through
    a controlchannel or RPC. The value indicates the state
    of the subscription as established by the publisher.";
}

leaf last-sequence-number {
  type yang:zero-based-counter64;
  config false;
  description
    "The envelope sequence number for the last notification
    sent for this subscription.

    The sequence number is initialized to zero when the
    subscription first becomes active and the first
    subscription-started notification is sent.";
}

leaf last-notification-time {
  type yang:date-and-time;
  config false;
  description
    "The notification envelope event-time timestamp of the
```



```
    last notification sent for this subscription.

    The timestamp is initialized to the time when the
    subscription first becomes active and the first
    subscription-started notification is sent.";
}

leaf last-periodic-collection-time {
    type yang:date-and-time;
    config false;
    description
        "The event-time timestamp of the last completed periodic
        collection or resynchronization collection for this
        subscription, and used as the event-time in the
        notification header.

        The timestamp is initialized to the time when the
        subscription first becomes active and the first
        subscription-started notification is sent.";
}

leaf last-on-change-notification-time {
    type yang:date-and-time;
    config false;
    description
        "The notification envelope event-time timestamp of the
        last on-change notification sent for this subscription.

        The timestamp is initialized to the time when the
        subscription first becomes active and the first
        subscription-started notification is sent.";
}

container statistics {
    config false;
    description
        "Statistics related to the number of messages generated
        for this subscription.";

    leaf started-notification-count {
        type yang:zero-based-counter64;
        config false;
        description
            "The number of subscription-started notifications
            sent for this subscription since the subscription
            list entry was created and the configuration
            was applied by the publisher.";
    }
}
```

```
leaf terminated-notification-count {
  type yang:zero-based-counter64;
  config false;
  description
    "The number of subscription-terminated notifications
    sent for this subscription since the subscription
    list entry was created and the configuration
    was applied by the publisher.";
}

leaf update-notification-count {
  type yang:zero-based-counter64;
  config false;
  description
    "The number of update records generated for the
    subscription, to be queued to one of more active
    receivers.

    The count is re-initialized to 0 when the
    subscription first becomes active and the
    subscription-started notification is sent.

    The count is incremented even if the update
    record has been generated, but is not queued to
    any receiver.";
}

leaf periodic-collection-count {
  type yang:zero-based-counter64;
  config false;
  description
    "The number of periodic collections completed for
    the subscription.

    The count is re-initialized to 0 when the
    subscription first becomes active and the
    subscription-started notification is sent.

    The count is incremented even if the update
    record has been generated, but is not queued to
    any receiver.";
}

leaf excluded-event-records {
  type yang:zero-based-counter64;
  config false;
  description
    "The number of event records explicitly removed via
```

```

        either an event stream filter or an access control
        filter so that they are not passed to a receiver.
        This count is set to zero each time
        'sent-event-records' is initialized.";
    }
}

action reset {
    description
        "Reset the subscription.

        This action will cause a new subscription-started
        message to be sent for the subscription";
}
}
}
container receivers {
    description
        "A container for all instances of configured receivers.";

    list receiver {
        key "name";

        leaf name {
            type string;
            description
                "An arbitrary but unique name for this receiver
                instance.";
        }

        leaf encoding {
/*            when 'not(..../transport) or derived-from(..../transport,
               "ypl:configurable-encoding")';*/
            type ypl:encoding;
            mandatory true;
            description
                "The type of encoding for notification messages.  For a
                dynamic subscription, if not included as part of an
                'establish-subscription' RPC, the encoding will be
                populated with the encoding used by that RPC.  For a
                configured subscription, if not explicitly configured,
                the encoding will be the default encoding for an
                underlying transport.";
        }
    }

    uses ypl:dscp;

    action reset {

```

```
description
  "Resets all configured subscriptions that reference
  this receiver.

  This action is directly equivalent to invoking the
  'reset' action on all subscriptions that references
  this receiver configuration.";
}

choice notification-message-origin {
  description
    "Identifies the egress interface on the publisher
    from which notification messages are to be sent.";
  case interface-originated {
    description
      "When notification messages are to egress a specific,
      designated interface on the publisher.";
    leaf source-interface {
      if-feature "interface-designation";
      type if:interface-ref;
      description
        "References the interface for notification
        messages.";
    }
  }
  case address-originated {
    description
      "When notification messages are to depart from a
      publisher using a specific originating address and/or
      routing context information.";
    leaf source-vrf {
      if-feature "supports-vrf";
      type leafref {
        path
          '/ni:network-instances/ni:network-instance/'
          + 'ni:name';
      }
      description
        "VRF from which notification messages should egress a
        publisher.";
    }
    leaf source-address {
      type inet:ip-address-no-zone;
      description
        "The source address for the notification messages.
        If a source VRF exists but this object doesn't, a
        publisher's default address for that VRF must
        be used.";
    }
  }
}
```

```
    }  
  }  
}  
  
choice transport-type {  
  mandatory true;  
  description  
    "Choice of different types of transports used to  
    send notifications. The 'case' statements must  
    be augmented in by other modules.";  
}  
  
description  
  "A list of all receiver instances.";  
}  
}  
}  
  
augment "/ypl:establish-subscription/ypl:input/ypl:"  
  + "target/ypl:filter" {  
  
  description  
    "Augment the 'establish-subscription' RPC to allow  
    referencing a configured receiver.";  
  
  case by-reference {  
    description  
      "Incorporates a filter that has been configured  
      separately.";  
    uses filter-ref {  
      refine "filter-ref" {  
        mandatory false;  
      }  
    }  
  }  
}  
  
augment "/ypl:subscription-started/ypl:target" {  
  
  description  
    "Allow a subscription-started notification to include a  
    referenced named filter";  
  uses filter-ref {  
    refine "filter-ref" {  
      mandatory false;  
    }  
  }  
}  
}
```

```

augment "/datastore-telemetry/subscriptions/subscription/"
  + "target/filter" {

  description
    "Augment the subscription filter to allow
     referencing a filter configured separately.";

  case by-reference {
    description
      "Incorporates a filter that has been configured
       separately.";
    uses filter-ref;
  }
}
}
<CODE ENDS>

```

Figure 22: YANG module ietf-yp-lite-config

14. Capabilities YANG Data Model

14.1. ietf-yp-lite-capabilities YANG tree

This section shows the tree output for ietf-yp-lite-capabilities YANG module, which augments the ietf-system-capabilities YANG module [RFC9196].

14.2. ietf-yp-lite-capabilities YANG Model

This module imports typedefs from the yang-push-lite YANG module.

This module augments the ietf-system-capabilities YANG module [RFC9196].

```

<CODE BEGINS> file "ietf-yp-lite-capabilities.yang#0.1.0"
module ietf-yp-lite-capabilities {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yp-lite-capabilities";
  prefix yplca;

  import ietf-system-capabilities {
    prefix sysc;
    reference
      "RFC 9196: YANG Modules Describing Capabilities for Systems
       and Datastore Update Notifications";
  }
  import ietf-yp-lite {
    prefix ypl;
  }
}

```

```
reference
  "RFC XXX: YANG Push Lite";
}

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

  Author:   Robert Wilton
            <mailto:rwilton@cisco.com>";
description
  "This module contains YANG specifications for YANG-Push lite.

  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) 2025 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 2024-11-11 {
  description
    "Initial revision.";
  reference
    "XXX: YANG Push Lite";
}

/*
 * IDENTITIES
 */

identity security-protocol {
  description
```

```
    "Identity for security protocols.";
}

identity tls12 {
  base security-protocol;
  description
    "Indicates TLS Protocol Version 1.2. TLS 1.2 is obsolete,
     and thus it is NOT RECOMMENDED to enable this feature.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
     Version 1.2";
}

identity tls13 {
  base security-protocol;
  description
    "Indicates TLS Protocol Version 1.3.";
  reference
    "RFC 8446: The Transport Layer Security (TLS)
     Protocol Version 1.3";
}

identity dtls12 {
  base security-protocol;
  description
    "Indicates DTLS Protocol Version 1.2. TLS 1.2 is obsolete,
     and thus it is NOT RECOMMENDED to enable this feature.";
  reference
    "RFC 6347: The Datagram Transport Layer Security (TLS) Protocol
     Version 1.2";
}

identity dtls13 {
  base security-protocol;
  description
    "Indicates DTLS Protocol Version 1.3.";
  reference
    "RFC 9147: The Datagram Transport Layer Security (TLS)
     Protocol Version 1.3";
}

identity ssh {
  base security-protocol;
  description
    "Indicates SSH.";
}
```



```
grouping yp-lite-capabilities {
  description
    "Capabilities related to YANG Push Lite subscriptions
    and notifications";
  container datastore-telemetry {
    description
      "Capabilities related to YANG Push List subscriptions
      and notifications";
    typedef notification-support {
      type bits {
        bit config-changes {
          description
            "The publisher is capable of sending
            notifications for 'config true' nodes for the
            relevant scope and subscription type.";
        }
        bit state-changes {
          description
            "The publisher is capable of sending
            notifications for 'config false' nodes for the
            relevant scope and subscription type.";
        }
      }
    }
    description
      "Type for defining whether 'on-change' or
      'periodic' notifications are supported for all data nodes,
      'config false' data nodes, 'config true' data nodes, or
      no data nodes.

      The bits config-changes or state-changes have no effect
      when they are set for a datastore or for a set of nodes
      that does not contain nodes with the indicated config
      value. In those cases, the effect is the same as if no
      support was declared. One example of this is indicating
      support for state-changes for a candidate datastore that
      has no effect."
  }
}

leaf periodic-notifications-supported {
  type notification-support;
  description
    "Specifies whether the publisher is capable of
    sending 'periodic' notifications for the selected
    data nodes, including any subtrees that may exist
    below them.";
  reference
    "RFC 8641: Subscription to YANG Notifications for
    Datastore Updates, 'periodic' subscription concept";
}
```

```
}
choice update-period {
  description
    "Supported update period value or values for
    'periodic' subscriptions.";
  leaf minimum-update-period {
    type uint32;
    units "centiseconds";
    description
      "Indicates the minimal update period that is
      supported for a 'periodic' subscription.

      A subscription request to the selected data nodes with
      a smaller period than what this leaf specifies is
      likely to result in a 'period-unsupported' error.";
  }
  leaf-list supported-update-period {
    type uint32;
    units "centiseconds";
    description
      "Supported update period values for a 'periodic'
      subscription.

      A subscription request to the selected data nodes with a
      period not included in the leaf-list will result in a
      'period-unsupported' error.";
  }
}
leaf on-change-supported {
  type notification-support;
  description
    "Specifies whether the publisher is capable of
    sending 'on-change' notifications for the selected
    data nodes and the subtree below them.";
}
}
}

grouping yp-lite-transport-capabilities {
  description
    "Capabilities related to transports supporting Yang Push Lite";
  container transport {
    description
      "Specifies capabilities related to YANG-Push transports.";
    list transport-capability {
      key "transport-protocol";
      description
        "Indicates a list of capabilities related to notification
```

```
        transport.";
    leaf transport-protocol {
        type identityref {
            base ypl:transport;
        }
        description
            "Indicates supported transport protocol for YANG-Push.";
    }
    leaf security-protocol {
        type identityref {
            base security-protocol;
        }
        description
            "Indicates transport security protocol.";
    }
    leaf-list encoding-format {
        type identityref {
            base ypl:encoding;
        }
        description
            "Indicates supported encoding formats.";
    }
}
}
}

// YANG Push Lite Capabilities
augment "/sysc:system-capabilities" {
    description
        "Adds system level capabilities for YANG Push Lite";
    uses yp-lite-capabilities;
}

augment "/sysc:system-capabilities/ypzca:datastore-telemetry" {
    description
        "Adds system level Yang Push Lite transport capabilities";
    uses yp-lite-transport-capabilities;
}

augment "/sysc:system-capabilities/sysc:datastore-capabilities"
    + "/sysc:per-node-capabilities" {
    description
        "Add datastore and node-level capabilities";
    uses yp-lite-capabilities;
}
}
<CODE ENDS>
```

Figure 23: YANG module ietf-yp-lite-capabilities

```

module: ietf-yp-lite-capabilities

augment /sysc:system-capabilities:
  +--ro datastore-telemetry
    +--ro periodic-notifications-supported?    notification-support
    +--ro (update-period)?
      | +--:(minimum-update-period)
      | | +--ro minimum-update-period?        uint32
      | +--:(supported-update-period)
      | | +--ro supported-update-period*       uint32
    +--ro on-change-supported?                notification-support
    +--ro transport
      +--ro transport-capability* [transport-protocol]
        +--ro transport-protocol              identityref
        +--ro security-protocol?              identityref
        +--ro encoding-format*                 identityref
  augment /sysc:system-capabilities/sysc:datastore-capabilities
    /sysc:per-node-capabilities:
      +--ro datastore-telemetry
        +--ro periodic-notifications-supported?    notification-support
        +--ro (update-period)?
          | +--:(minimum-update-period)
          | | +--ro minimum-update-period?        uint32
          | +--:(supported-update-period)
          | | +--ro supported-update-period*       uint32
        +--ro on-change-supported?                notification-support

```

Figure 24: YANG tree for YANG Push Lite Capabilities Module Tree Output

15. Security Considerations

With configured subscriptions, one or more publishers could be used to overwhelm a receiver. To counter this, notification messages SHOULD NOT be sent to any receiver that does not support this specification. Receivers that do not want notification messages need only terminate or refuse any transport sessions from the publisher.

When a receiver of a configured subscription gets a new "subscription-started" message for a known subscription where it is already consuming events, it may indicate that an attacker has done something that has momentarily disrupted receiver connectivity. *TODO - Do we still want this paragraph?*

For dynamic subscriptions, implementations need to protect against malicious or buggy subscribers that may send a large number of "establish-subscription" requests and thereby use up system resources. To cover this possibility, operators SHOULD monitor for such cases and, if discovered, take remedial action to limit the resources used, such as suspending or terminating a subset of the subscriptions or, if the underlying transport is session based, terminating the underlying transport session.

Using DNS names for configured subscription's receiver "name" lookups can cause situations where the name resolves differently than expected on the publisher, so the recipient would be different than expected.

15.1. Use of YANG Push Lite with NACM

TODO, do we even need this section?

This specification MAY be used with access control tools, such as NACM [RFC8341]. Please refer to that specification for normative guidance of how NACM applies.

For informative purposes, please note that NACM can be used:

- * NACM can be used to control access to the data nodes and RPCs defined in the YANG modules defined in this document.
- * NACM can be used to control access to the data included in the YANG _update_ notifications.

15.2. Receiver Authorization

TODO Relax when access control must be checked.

TODO Consider if this is the best place in the document, but this text needs to be updated regardless.

A receiver of subscription data MUST only be sent updates for which it has proper authorization. A publisher MUST ensure that no unauthorized data is included in push updates. To do so, it needs to apply all corresponding checks applicable at the time of a specific pushed update and, if necessary, silently remove any unauthorized data from datastore subtrees. This enables YANG data that is pushed based on subscriptions to be authorized in a way that is equivalent to a regular data retrieval ("get") operation.

Each "push-update" and "push-change-update" MUST have access control applied, as depicted in Figure 5. This includes validating that read access is permitted for any new objects selected since the last notification message was sent to a particular receiver. A publisher MUST silently omit data nodes from the results that the client is not authorized to see. To accomplish this, implementations SHOULD apply the conceptual authorization model of [RFC8341], specifically Section 3.2.4, extended to apply analogously to data nodes included in notifications, not just <rpc-reply> messages sent in response to <get> and <get-config> requests.

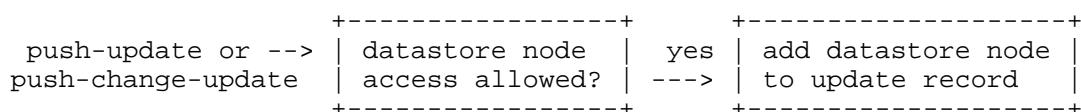


Figure 25: Access Control for Push Updates

A publisher MUST allow for the possibility that a subscription's selection filter references nonexistent data or data that a receiver is not allowed to access. Such support permits a receiver the ability to monitor the entire lifecycle of some datastore tree without needing to explicitly enumerate every individual datastore node. If, after access control has been applied, there are no objects remaining in an update record, then the effect varies given if the subscription is a periodic or on-change subscription. For a periodic subscription, an empty "push-update" notification MUST be sent, so that clients do not get confused into thinking that an update was lost. For an on-change subscription, a "push-update" notification MUST NOT be sent, so that clients remain unaware of changes made to nodes they don't have read-access for.

A publisher MAY choose to reject an "establish-subscription" request that selects nonexistent data or data that a receiver is not allowed to access. The error identity "unchanging-selection" SHOULD be returned as the reason for the rejection. In addition, a publisher MAY choose to terminate a dynamic subscription or suspend a configured receiver when the authorization privileges of a receiver change or the access controls for subscribed objects change. In that case, the publisher SHOULD include the error identity "unchanging-selection" as the reason when sending the "subscription-terminated" or "subscription-suspended" notification, respectively. Such a capability enables the publisher to avoid having to support continuous and total filtering of a subscription's content for every update record. It also reduces the possibility of leakage of access-controlled objects.

If read access into previously accessible nodes has been lost due to a receiver permissions change, this SHOULD be reported as a patch "delete" operation for on-change subscriptions. If not capable of handling such receiver permission changes with such a "delete", publisher implementations MUST force dynamic subscription re-establishment or configured subscription reinitialization so that appropriate filtering is installed.

15.3. YANG Module Security Considerations

This section is modeled after the template described in Section 3.7.1 of [I-D.draft-ietf-netmod-rfc8407bis].

The "ietf-yp-lite" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These protocols have to use a secure transport layer (e.g., SSH [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and have to use mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a pre-configured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., "config true", which is the default). All writable data nodes are likely to be reasonably sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) and delete operations to these data nodes without proper protection or authentication can have a negative effect on network operations. The following subtrees and data nodes have particular sensitivities/vulnerabilities:

- * There are no particularly sensitive writable data nodes.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. Specifically, the following subtrees and data nodes have particular sensitivities/vulnerabilities:

- * There are no particularly sensitive readable data nodes.

Some of the RPC or action operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. Specifically, the following operations have particular sensitivities/vulnerabilities:

- * `kill-subscription` - this RPC operation allows the caller to kill any dynamic subscription, even those created via other users, or other transport sessions.

16. IANA Considerations

16.1. Namespace URI registrations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

=====+		=====+
	URI	
=====+		=====+
	<code>urn:ietf:params:xml:ns:yang:ietf-yp-lite</code>	
+-----+		+-----+
	<code>urn:ietf:params:xml:ns:yang:ietf-yp-lite-config</code>	
+-----+		+-----+
	<code>urn:ietf:params:xml:ns:yang:ietf-yp-lite-capabilities</code>	
+-----+		+-----+

Table 1: Namespace URI registrations

For all registrations:

- * Registrant Contact: The IESG.
- * XML: N/A; the requested URI is an XML namespace.

17. YANG Module Name registrations

This document registers the following YANG modules in the "YANG Module Names" registry [RFC6020]:

Name	Namespace	Prefix
ietf-yp-lite	urn:ietf:params:xml:ns:yang:ietf-yp-lite	ypl
ietf-yp-lite-config	urn:ietf:params:xml:ns:yang:ietf-yp-lite-config	yplco
ietf-yp-lite-capabilities	urn:ietf:params:xml:ns:yang:ietf-yp-lite-capabilities	yplca

Table 2: YANG Module Name Registrations

For all registration the reference is "RFC XXXX".

Acknowledgments

This initial draft is early work is based on discussions with various folk, particularly Thomas Graf, Holger Keller, Dan Voyer, Nils Warnke, and Alex Huang Feng; but also wider conversations that include: Benoit Claise, Pierre Francois, Paolo Lucente, Jean Quilbeuf, among others.

Contributors

The following individuals have actively contributed to this draft and the YANG Push Solution.

* Dan Voyer

References

Normative References

- [BCP14] 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>>.
- [I-D.draft-ietf-netconf-notif-envelope] Feng, A. H., Francois, P., Graf, T., and B. Claise, "Extensible YANG Model for YANG-Push Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-notif-envelope-03, 20 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-notif-envelope-03>>.

- [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>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/rfc/rfc2474>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/rfc/rfc6241>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/rfc/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/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>>.
- [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>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/rfc/rfc8341>>.
- [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/rfc/rfc8342>>.

- [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/rfc/rfc8525>>.
- [RFC8529] Berger, L., Hopps, C., Lindem, A., Bogdanovic, D., and X. Liu, "YANG Data Model for Network Instances", RFC 8529, DOI 10.17487/RFC8529, March 2019, <<https://www.rfc-editor.org/rfc/rfc8529>>.
- [RFC8791] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <<https://www.rfc-editor.org/rfc/rfc8791>>.
- [RFC9196] Lengyel, B., Clemm, A., and B. Claise, "YANG Modules Describing Capabilities for Systems and Datastore Update Notifications", RFC 9196, DOI 10.17487/RFC9196, February 2022, <<https://www.rfc-editor.org/rfc/rfc9196>>.
- [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>>.
- [RFC9485] Bormann, C. and T. Bray, "I-Regexp: An Interoperable Regular Expression Format", RFC 9485, DOI 10.17487/RFC9485, October 2023, <<https://www.rfc-editor.org/rfc/rfc9485>>.
- [RFC9595] Veillette, M., Ed., Pelov, A., Ed., Petrov, I., Ed., Bormann, C., and M. Richardson, "YANG Schema Item Identifier (YANG SID)", RFC 9595, DOI 10.17487/RFC9595, July 2024, <<https://www.rfc-editor.org/rfc/rfc9595>>.

Informative References

- [Consistency] Wikipedia, "Consistency (database systems)", <[https://en.wikipedia.org/wiki/Consistency_\(database_systems\)](https://en.wikipedia.org/wiki/Consistency_(database_systems))>.
- [EventualConsistency] Rouse, M., "Eventual Consistency", <<https://www.techopedia.com/definition/29165/eventual-consistency>>.

- [gNMI] OpenConfig, "gRPC Network Management Interface (gNMI)", <<https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md>>.
- [I-D.draft-ietf-netconf-distributed-notif]
Zhou, T., Zheng, G., Voit, E., Graf, T., and P. Francois, "Subscription to Notifications in a Distributed Architecture", Work in Progress, Internet-Draft, draft-ietf-netconf-distributed-notif-16, 19 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-distributed-notif-16>>.
- [I-D.draft-ietf-netconf-https-notif]
Jethanandani, M. and K. Watsen, "An HTTPS-based Transport for YANG Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-https-notif-15, 1 February 2024, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-15>>.
- [I-D.draft-ietf-netconf-udp-notif]
Feng, A. H., Francois, P., Zhou, T., Graf, T., and P. Lucente, "UDP-based Transport for Configured Subscriptions", Work in Progress, Internet-Draft, draft-ietf-netconf-udp-notif-23, 14 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-udp-notif-23>>.
- [I-D.draft-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-28, 5 June 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-netmod-rfc8407bis-28>>.
- [I-D.draft-netana-netconf-yp-transport-capabilities]
Wu, Q., Ma, Q., Feng, A. H., and T. Graf, "YANG Notification Transport Capabilities", Work in Progress, Internet-Draft, draft-netana-netconf-yp-transport-capabilities-01, 17 January 2025, <<https://datatracker.ietf.org/doc/html/draft-netana-netconf-yp-transport-capabilities-01>>.

[I-D.ietf-netconf-distributed-notif]

Zhou, T., Zheng, G., Voit, E., Graf, T., and P. Francois, "Subscription to Notifications in a Distributed Architecture", Work in Progress, Internet-Draft, draft-ietf-netconf-distributed-notif-16, 19 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-distributed-notif-16>>.

[I-D.ietf-nmop-network-anomaly-architecture]

Graf, T., Du, W., Francois, P., and A. H. Feng, "A Framework for a Network Anomaly Detection Architecture", Work in Progress, Internet-Draft, draft-ietf-nmop-network-anomaly-architecture-05, 6 September 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-nmop-network-anomaly-architecture-05>>.

[I-D.ietf-nmop-yang-message-broker-integration]

Graf, T. and A. Elhassany, "An Architecture for YANG-Push to Message Broker Integration", Work in Progress, Internet-Draft, draft-ietf-nmop-yang-message-broker-integration-09, 20 October 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-nmop-yang-message-broker-integration-09>>.

[I-D.tgraf-netconf-notif-sequencing]

Graf, T., Quilbeuf, J., and A. H. Feng, "Support of Hostname and Sequencing in YANG Notifications", Work in Progress, Internet-Draft, draft-tgraf-netconf-notif-sequencing-06, 29 June 2024, <<https://datatracker.ietf.org/doc/html/draft-tgraf-netconf-notif-sequencing-06>>.

[I-D.tgraf-netconf-yang-push-observation-time]

Graf, T., Claise, B., and A. H. Feng, "Support of Observation Timestamp in YANG-Push Notifications", Work in Progress, Internet-Draft, draft-tgraf-netconf-yang-push-observation-time-03, 14 December 2024, <<https://datatracker.ietf.org/doc/html/draft-tgraf-netconf-yang-push-observation-time-03>>.

[Kafka] Apache.org, "Apache Kafka", <<https://kafka.apache.org/>>.

[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/rfc/rfc3411>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/rfc/rfc3688>>.
- [RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <<https://www.rfc-editor.org/rfc/rfc4252>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<https://www.rfc-editor.org/rfc/rfc5277>>.
- [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>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/rfc/rfc7049>>.
- [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/rfc/rfc7540>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/rfc/rfc8040>>.
- [RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <<https://www.rfc-editor.org/rfc/rfc8071>>.
- [RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <<https://www.rfc-editor.org/rfc/rfc8072>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/rfc/rfc8259>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/rfc/rfc8343>>.

- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/rfc/rfc8446>>.
- [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/rfc/rfc8639>>.
- [RFC8640] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Dynamic Subscription to YANG Events and Datastores over NETCONF", RFC 8640, DOI 10.17487/RFC8640, September 2019, <<https://www.rfc-editor.org/rfc/rfc8640>>.
- [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>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/rfc/rfc9000>>.
- [XPath] W3C, "XML Path Language (XPath) Version 1.0", <<https://www.w3.org/TR/1999/REC-xpath-19991116/>>.

Appendix A. Functional changes between YANG Push Lite and YANG Push

This non-normative section highlights the significant functional changes where the YANG Push Lite implementation differs from YANG Push. However, the main body of this document, from Section 4 onwards, provides the normative definition of the YANG Push Lite specification, except for any text or sections that explicitly indicate that they are informative rather being normative.

Note to reviewers: If you notice mistakes in this section during development of the document and solution then please point them out to the authors and the working group. _*(RFC editor, please remove this paragraph prior to publication)*

A.1. Removed Functionality

This section lists functionality specified in [RFC8639] and YANG Push which is not specified in YANG Push Lite.

- * Negotiation and hints of failed subscriptions.

- * The RPC to modify an existing dynamic subscription, instead the subscription must be terminated and re-established.
- * The ability to suspend and resume a dynamic subscription. Instead a dynamic subscription is terminated if the device cannot reliably fulfill the subscription or a receiver is too slow causing the subscription to be back pressured.
- * Specifying a subscription stop time, and the corresponding subscription-completed notification have been removed.
- * Replaying of buffered event records are not supported. The nearest equivalent is requesting a sync-on-start replay when the subscription transport session comes up which will reply the current state.
- * QoS weighting and dependency between subscriptions has been removed due to the complexity of implementation.
- * Support for reporting subscription error hints has been removed. The device SHOULD reject subscriptions that are likely to overload the device, but more onus is placed on the operator configuring the subscriptions or setting up the dynamic subscriptions to ensure that subscriptions are reasonable, as they would be expected to do for any other configuration.
- * The "subscription-state-notif" extension has been removed.
- * The YANG Patch format [RFC8072] is no longer used for on-change subscriptions.

A.2. Changed Functionality

This section documents behavior that exists in both YANG Push and YANG Push Lite, but the behavior differs between the two:

- * All YANG Push Lite notifications messages use [I-D.draft-ietf-netconf-notif-envelope] rather than [RFC5277] used by YANG Push [RFC8641].
- * Changes to handling receivers:
 - Receivers are always configured separately from the subscription and are referenced.
 - Transport and Encoding parameters are configured as part of a receiver definition, and are used by all subscriptions directed towards a given receiver.

- Encoding is now a mandatory parameter under a receiver and dynamic subscription (rather than specifying a default).
- If a subscription uses multiple receivers then:
 - o Update messages and lifecycle notifications are sent to all receivers (to preserve sequence numbers)
 - o all receivers must be configured with the same encoding
- Invoking the `_reset_` RPC operation on a receiver requires and forces a reset of any transport sessions associated with that receiver. Previously, the sessions would not be reset if they were used by other subscriptions.
- * Periodic and on-change message uses a common `_update_` notification message format, allowing for the messages to be processed by clients in a similar fashion and to support combined periodic and on-change subscriptions.
- * On-change dampening:
 - Client configurable on-change dampening has been removed.
 - However, YANG Push Lite allows a publisher to limit the rate at which a data node is sampled for on-change notifications. See Section 7.5.1 for further details.
- * Dynamic subscriptions are no longer mandatory to implement, either or both of Configured and Dynamic Subscriptions may be implemented in YANG Push Lite.
- * The solution focuses solely on datastore subscriptions that each have their own event stream. Filters cannot be applied to the event stream, only to the set of datastore data nodes that are monitored by the subscription.
- * The lifecycle events of when a subscription-started or subscription-terminated may be sent differs from RFC 8639/RFC 8649:
 - Subscription-started notifications are also sent for dynamic subscriptions.
- * Some of the requirements on transport have been relaxed.

- * The encoding identities have been extended with CBOR encodings, and the "encoding-" prefix has been removed (so that there is a better on the wire representation).
- * YANG Push Lite allows for a publisher to provide an eventually consistent distributed view of the operational datastore, rather than a fully consistent datastore where on-change updates are sent as logic diffs to that datastore.

A.3. Added Functionality

- * Device capabilities are reported via XXX and additional models that augment that data model.
- * A new `_update_` message:
 - Covers both on-change and periodic events.
 - Allows multiple updates to be sent in a single message (e.g., for on-change).
 - Allows for a common path prefix to be specified, with any paths and encoded YANG data to be encoded relative to the common path prefix.
- * A `_collection-complete_` notification, and associated configuration, has been defined to inform collectors when a subscription's periodic collection cycle is complete.
- * TODO - More operational data on the subscription load and performance.
- * All YANG Push Lite configuration is under a new `_datastore-telemetry_` presence container

Appendix B. Subscription Errors (from RFC 8641)

B.1. RPC Failures

Rejection of an RPC for any reason is indicated via an RPC error response from the publisher. Valid RPC errors returned include both (1) existing transport-layer RPC error codes, such as those seen with NETCONF in [RFC6241] and (2) subscription-specific errors, such as those defined in the YANG data model. As a result, how subscription errors are encoded in an RPC error response is transport dependent.

References to specific identities in the ietf-subscribed-notifications YANG module [RFC8639] or the ietf-yang-push YANG module may be returned as part of the error responses resulting from failed attempts at datastore subscription. For errors defined as part of the ietf-subscribed-notifications YANG module, please refer to [RFC8639]. The errors defined in this document, grouped per RPC, are as follows:

establish-subscription	modify-subscription
-----	-----
cant-exclude	period-unsupported
datastore-not-subscribable	update-too-big
on-change-unsupported	sync-too-big
on-change-sync-unsupported	unchanging-selection
period-unsupported	
update-too-big	resync-subscription
sync-too-big	-----
unchanging-selection	no-such-subscription-resync
	sync-too-big

There is one final set of transport-independent RPC error elements included in the YANG data model. These are the four yang-data structures for failed datastore subscriptions:

1. yang-data "establish-subscription-error-datastore": This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere in the transport portion of a failed "establish-subscription" RPC response. This MUST be sent if hints are included.
2. yang-data "modify-subscription-error-datastore": This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere in the transport portion of a failed "modify-subscription" RPC response. This MUST be sent if hints are included.
3. yang-data "sn:delete-subscription-error": This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere in the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.
4. yang-data "resync-subscription-error": This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere in the transport portion of a failed "resync-subscription" RPC response.

B.2. Failure Notifications

A subscription may be unexpectedly terminated or suspended independently of any RPC or configuration operation. In such cases, indications of such a failure MUST be provided. To accomplish this, a number of errors can be returned as part of the corresponding subscription state change notification. For this purpose, the following error identities are introduced in this document, in addition to those that were already defined in [RFC8639]:

subscription-terminated	subscription-suspended
-----	-----
datastore-not-subscribable	period-unsupported
unchanging-selection	update-too-big
	synchronization-size

Appendix C. Examples

Notes on examples:

- * To allow for programmatic validation, most notification examples in this section exclude the mandatory notification envelope and associated metadata defined in [I-D.draft-ietf-netconf-notif-envelope]. Only the full notification example in Section 7.1 includes the notification header.
- * These notification message examples are given using a JSON encoding, but could be encoded using XML or CBOR.
- * Some additional meta data fields, e.g., like those defined in [I-D.tgraf-netconf-notif-sequencing] would also likely be included, but have also been excluded to allow for slightly more concise examples.
- * The examples include the [I-D.tgraf-netconf-yang-push-observation-time] field for the existing YANG-Push Notification format, and the proposed equivalent "observation-time" leaf for the new update notification format.
- * All these examples are created by hand, may contain errors, and may not parse correctly.

C.1. Example of periodic update messages

In this example, a subscription is for `_/ietf-interfaces:interfaces/interface_`. However, for efficiency reasons, the publisher is internally returning the data from two different data providers.

Of note:

- * The first periodic message is published for the entries in the `_/ietf-interfaces:interface/interfaces_ list`, but doesn't contain the data in the `_statistics_ child` container.
- * the `_path-prefix_` is to the subscription subtree, since the device will never return data outside of the subscription subtree.
- * the `_target-path_` is elided because the data is returned at the subscription point. ****TODO, or should it actually be to the element above? ****

```

{
  "ietf-yp-lite:update": {
    "id": 1011,
    "path-prefix": "/ietf-interfaces:interfaces/interface",
    "snapshot-type": "periodic",
    "observation-time": "2024-10-10T08:00:05.11Z",
    "updates": [
      {
        "target-path": "",
        "data": {
          "ietf-interfaces:interface": [
            {
              "name": "eth0",
              "type": "iana-if-type:ethernetCsmacd",
              "enabled": true,
              "ietf-interfaces:oper-status": "up",
              "ietf-interfaces:admin-status": "up"
            },
            {
              "name": "eth1",
              "type": "iana-if-type:ethernetCsmacd",
              "enabled": true,
              "ietf-interfaces:oper-status": "up",
              "ietf-interfaces:admin-status": "up"
            }
          ]
        }
      }
    ],
    "complete": [null]
  }
}

```

Figure 26: Example periodic update for interfaces list

For the second notification related to the same subscription:

- * the second periodic message is published for only the statistics associated with the interfaces.
- * as above, the `_path-prefix_` is still to the subscription subtree.
- * the second notification uses a separate observation-time, but would use the same event-time in the notification header so that the two messages can be correlated to the same periodic collection event.

- * the second periodic message has set the `_complete_` flag to indicate that it is the last notification as part of the periodic collection. A separate `_update-complete_` notification could have been sent instead.

```
{
  "ietf-yp-lite:update": {
    "id": 1011,
    "path-prefix": "/ietf-interfaces:interfaces/interface",
    "snapshot-type": "periodic",
    "observation-time": "2024-10-10T08:00:05.21Z",
    "updates": [
      {
        "target-path": "",
        "data": {
          "ietf-interfaces:interface": [
            {
              "name": "eth0",
              "statistics": {
                "in-octets": 123456,
                "out-octets": 654321
              }
            },
            {
              "name": "eth1",
              "statistics": {
                "in-octets": 789012,
                "out-octets": 210987
              }
            }
          ]
        }
      }
    ]
  }
}
```

Figure 27: Example periodic update for interface statistics

C.2. Example of an on-change-update notification using the new style update message

If the subscription is for on-change notifications, or periodic-and-on-change-notifications, then, if the interface state changed (specifically if the 'state' leaf of the interface changed state), and if the device was capable of generating on-change notifications, then you may see the following message. A few points of notes here:

- * The on-change notification contains **all** of the state at the "target-path"
 - Not present in the below example, but if the notification excluded some state under an interfaces list entry (e.g., the line-state leaf) then this would logically represent the implicit deletion of that field under the given list entry.
 - In this example it is restricted to a single interface. It could also publish an on-change notification for all interfaces, by indicating a target-path without any keys specified. TODO - Can it represent notifications for a subset of interfaces?
- * The schema of the change message is exactly the same as for the equivalent periodic message. It doesn't use the YANG Patch format [RFC8072] for on-change messages.
- * The "observation time" leaf represents when the system first observed the on-change event occurring.
- * The on-change event doesn't differentiate the type of change to operational state. The on-change-update snapshot type is used to indicate the creation of a new entry or some update to some existing state. Basically, the message can be thought of as the state existing with some current value.


```

<CODE BEGINS> file "on-change-msg.json"
{
  "ietf-yp-notification:envelope": {
    "event-time": "2024-09-27T14:16:30.973Z",
    "hostname": "example-router",
    "sequence-number": 454,
    "contents": {
      "ietf-yp-ext:update": {
        "id": 1,
        "subscription-path":
          "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces",
        "target-path":
          "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interfaces/
          interface[interface=GigabitEthernet0/0/0/0]",
        "snapshot-type": "on-change-update"
        "observation-time": "2024-09-27T14:16:30.973Z",
        "datastore-snapshot": {
          "interfaces": {
            "interface": [
              {
                "interface-name": "GigabitEthernet0/0/0/0",
                "interface": "GigabitEthernet0/0/0/0",
                "state": "im-state-up",
                "line-state": "im-state-up"
              }
            ]
          }
        }
      }
    }
  }
}
<CODE ENDS>

```

Figure 28: Example YANG Push Lite on-change update message

C.3. Example of an on-change-delete notification using the new style update message

C.3.1. Update message with single deleted data node

If the interface was deleted, and if the system was capable of reporting on-change events for the delete event, then an on-change delete message would be encoded as per the following message.

Of note:

- * The ietf-yp-notification:envelope has been elided

- * The deleted data is identified by the target node in the `_updates/target-path_` element.
- * The observation time represents the time at which the delete event occurred, e.g., perhaps when the system processed a configuration change.

```

{
  "ietf-yp-lite:update": {
    "id": 1012,
    "snapshot-type": "on-change-delete",
    "observation-time": "2025-10-10T08:16:15.11Z",
    "updates": [
      {
        "target-path": "/ietf-interfaces:interfaces/interface[name='eth0']"
      }
    ]
  }
}

```

Figure 29: Example YANG Push Lite on-change delete message

C.3.2. Update message with multiple on-change deletes

This follow example illustrates how a single update notification message can contain multiple on-change delete events for different data nodes. In this example, two separate interfaces are being deleted.

Of note:

- * The `ietf-yp-notification:envelope` has been elided
- * `_prefix-path_` is used to shorten the target-paths, the full paths can be constructed concatenating the `_prefix-path_` with each `_target-path_` in the `_updates_` list.
- * all delete events share a common observation-time of when the delete events occurred. If it is necessary to identify separate observation times then the publisher would send separate messages.
- * data node subtrees that are deleted (list entries in this case) are identified by separate entries in the `_updates_` list.

```
{
  "ietf-yp-lite:update": {
    "id": 1013,
    "path-prefix": "/ietf-interfaces:interfaces",
    "snapshot-type": "on-change-delete",
    "observation-time": "2025-10-10T08:16:16.11Z",
    "updates": [
      {
        "target-path": "interface[name='eth0']"
      },
      {
        "target-path": "interface[name='eth1']"
      }
    ]
  }
}
```

Figure 30: Example YANG Push Lite on-change delete message

C.4. Subscription RPC examples (from RFC 8641)

YANG-Push subscriptions are established, modified, and deleted using RPCs augmented from [RFC8639].

C.4.1. "establish-subscription" RPC

The subscriber sends an "establish-subscription" RPC with the parameters listed in Section 3.1. An example might look like:

```

<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
    xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
      xmlns:ex="https://example.com/sample-data/1.0">
      /ex:foo
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
  </establish-subscription>
</netconf:rpc>

```

Figure 10: "establish-subscription" RPC

A positive response includes the "id" of the accepted subscription. In that case, a publisher may respond as follows:

```

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <id
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    52
  </id>
</rpc-reply>

```

Figure 11: "establish-subscription" Positive RPC Response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, no capacity to serve the subscription at the publisher, or the inability of the publisher to select datastore content at the requested cadence.

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error hints that help a subscriber understand what subscription parameters might have been accepted for the request. These hints would be included in the yang-data structure "establish-subscription-error-datastore". However, even with these hints, there are no guarantees that subsequent requests will in fact be accepted.

The specific parameters to be returned as part of the RPC error response depend on the specific transport that is used to manage the subscription. For NETCONF, those parameters are defined in [RFC8640]. For example, for the following NETCONF request:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns=
      "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
    xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
      xmlns:ex="https://example.com/sample-data/1.0">
      /ex:foo
    </yp:datastore-xpath-filter>
    <yp:on-change>
    </yp:on-change>
  </establish-subscription>
</rpc>
```

Figure 12: "establish-subscription" Request: Example 2

A publisher that cannot serve on-change updates but can serve periodic updates might return the following NETCONF response:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-path>/yp:periodic/yp:period</error-path>
    <error-info>
      <yp:establish-subscription-error-datastore>
        <yp:reason>yp:on-change-unsupported</yp:reason>
      </yp:establish-subscription-error-datastore>
    </error-info>
  </rpc-error>
</rpc-reply>
```

Figure 13: "establish-subscription" Error Response: Example 2

C.4.2. "delete-subscription" RPC

To stop receiving updates from a subscription and effectively delete a subscription that had previously been established using an "establish-subscription" RPC, a subscriber can send a "delete-subscription" RPC, which takes as its only input the subscription's "id". This RPC is unmodified from [RFC8639].

C.4.3. "resync-subscription" RPC

This RPC is supported only for on-change subscriptions previously established using an "establish-subscription" RPC. For example:

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <resync-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>1011</id>
  </resync-subscription>
</rpc>
```

Figure 15: "resync-subscription"

On receipt, a publisher must either (1) accept the request and quickly follow with a "push-update" or (2) send an appropriate error in an RPC error response. In its error response, the publisher MAY include, in the yang-data structure "resync-subscription-error", supplemental information about the reasons for the error.

Appendix D. Summary of Open Issues & Potential Enhancements

This temporary section lists open issues and enhancements that require further discussion by the authors, or the WG. Once adopted, it is anticipated that tracking the open issues would move to github.

The issues are ordered/grouped by the sections in the current document. I.e., to make it easier to review/update sections of the document.

D.1. Issues related to general IETF process

1. If this work progresses we will need simple bis versions of the transports document so that they augment into the new data model paths. Drafts that would need to be updated as documented in Section 3.4.

2. Do we need to fold in any text from RFC 8640? and RESTCONF.
I.e., there was this text in the one of the previous docs:
Bindings for subscribed event record delivery for NETCONF and
RESTCONF are defined in [RFC8640] and *RESTCONF-Notif*,
respectively.

* Rob: If text is needed for NETCONF and/or RESTCONF then I
suspect that it would be better added to this document than to
require small separate documents (as was done before).
3. What is the right name. Should we be calling this Yang Push 2
(YPv2) to more clearly indicate that this is intended to be a
replacement for Yang Push? We need to be slightly careful in
that this is only specifying implementing a subset of the
functionality.

D.2. Issue related to Terminology/Definitions

1. Should we use the object terminology?

* Holger: should try and use same terminology as existing YANG
Push RFCs.

* Rob: This issue can be a place holder, to go through and
check/update the terminology used, and see if any of the
referenced terminology isn't needed in YP Lite.

* Rob: Update, possible should refine the definition of object
to make it clear that it may refer to a single data node or
set of data nodes.

D.3. Issues related to YANG Push Lite Overview

None currently.

D.4. Issues related to Subscription Paths and Selection Filters

1. This draft introduces a new simple yang path (ypath) format that
is like a JSON instance data path, that all implementations MUST
support.

* Vendors already support a simple JSON like path (e.g., using
module-names rather than XML namespaces).

* *Open questions (for further discussion):*

- Do we support filtering on non-keys, e.g., this example from Thomas (but which filters on a non-key):
/if:interfaces-state/if:interface[if:type='ianaift:ethernet
Csmacd']/if:statistics
 - Do we support simple regex filtering on the keys (and if so, how would that be expressed, would it be compatible with JSON Path)
 - Do we need any more complex filtering (e.g., Holger's example of only getting entries from a list if they are in a particular state.), or do we always use subtree filters for that?
- * *Authors: I provides some suggested text in Section 6.1, but I expect further discussion may be needed. (E.g., how multiple keys should be expressed)*

2. Advertising subscription schema: <https://github.com/rgwilton/draft-yp-observability/issues/11>

D.5. Issues related to Datastore Event Streams

D.5.1. Further refinements on how a subscription can be decomposed internally into child subscriptions with the data returned for each child subscription:

1. Handling lists with separate producers of list entries.

- * If a subscription is decomposed, then should the subscription started message for the configured subscription indicate how that subscription has been decomposed?
 - * Do we need to add an optional replay-end (i.e., after a sync-on-start) or collection-end (i.e., after every collection) notification so that clients can determine when data can be implicitly deleted.
- Rob: I think that we should add the latter, but make it a subscription config option to turn it on.
 - Holger, strong support for adding this, often requested.
 - Thomas, yes we need this, but need to determine whether this is per subscription or per publisher.

D.5.2. Questions/comments on the notification message format:

1. We allow multiple updates within a single message (primary use case is for the on-change case). What about the timestamp, which is still just once per message (like gNMI)? Should message bundling be optional/configurable to implement (if they all use a single shared timestamp)?
 - * on-change deletes are implicit by an update that replaces an existing entry with an empty data node (e.g., "{}" in JSON)
 - An alternative choice here could be an explicit delete flag, we need to decide which would be simpler/better.
 - * The update message also currently includes a path-prefix to allow (like gNMI) so that they don't necessarily need to be encoded from root, specifically, I think that this makes on-change messages nicer, since the on-change is rooted to the thing that is changing rather than the root of the tree. We need to define semantics of how this works, e.g., this should probably be controlled via configuration.
 - Ideally, the path prefix would be exactly the same as the YPath format that is being specified for configuration.
 - Thomas: We should keep this.
 - * Further discussion of this issue is needed

D.6. Issues related to Receivers, Transports, & Encodings

D.6.1. Constraints on supporting multiple receivers:

1. Encoding is set per receiver, but do we need to support a subscription supporting more than one receiver with different encodings?
 - * Rob: Note, it is always possible for a client to setup different subscriptions.
 - * Thomas: This could be useful, e.g., when migrating from one encoding to another, particularly, if it reduced the load on the publisher (compared to creating a new separate subscription).
 - * Rob: I don't think that we would end up optimizing for this case (but would need to check).

- * Also need to check with Nokia/Huawei would support this.
 - * *Current status: Left open for further discussion.*
 - * Note: This may also have a minor impact on the data model.
2. Lifecycle message should be sent per subscription, not per receiver (i.e., every receiver gets the same message (ignoring transport headers)):
- * Consensus agreed this is the right thing to do.
 - * *Action on Rob to check/update the draft.*
 - * Notes:
 - Every receiver gets the same sequence number in the message.
 - subscription-created notification could perhaps have an enum giving a reason for the notification (e.g., new subscription, receiver added).
3. Do we need a per-receiver notification to indicate to a specific receiver that it has been disconnected?
- * This would seem to only apply to configured subscriptions (since dynamic cannot have more than one receiver).
 - * If the transport session went down, then the publisher would be expected to reconnect anyway.
 - * So, the specific use case is likely to be unconfiguring a receiver for a subscription that has multiple receivers.
 - * *Currently parked for further discussion.*

D.6.2. Issues related to Transports:

1. Section 8.2 lists quite a lot of rules on what are valid transports and what negotiation/etc is required. I think we need to check whether we can weaken some of these (although it is possible that these were imposed during a transport directorate review).
- * *Rob: Authors, I've updated the transport section, Section 8.2, please can you re-review.*

2. James: We also need to add some text into security section.
 - * Rob: Is this transport specific, or related to application layer authorization?
3. What is the rules/restrictions for subscription receiver instances vs transport sessions? E.g., is this entirely down to the transport to define.

D.7. Issues related to Setting up & Managing Subscriptions

D.7.1. Issues related to the configuration model:

Note some of these apply or impact dynamic subscriptions as well.

1. YP Lite is somewhat different (separate namespace, separate receivers, no event filters, some config has moved to a separate receivers list.) See the data model and Appendix A.
2. We should allow devices to limit which datastores subscriptions can be made against (e.g., not candidate or factory-default as some obvious examples). Should these be advertised in the capabilities?
3. (James) Should we even document the configuration data model in this document at all, or should it be in a separate document?
4. Some other changes/proposed changes:
 - * I've removed some of the YANG features for 'small' one/two changes in configuration. I propose that we don't need features for these at all (e.g., DSCP settings), or in other cases we can use operational capabilities, or rely on deviations.
 - I've renamed the encodings (e.g., from "encode-json" to just "json")
 - Maybe further simplification of the receivers list under the subscription. E.g., do we need stats per subscription per receiver, or just per subscription? Do want stats across all subscriptions to a given receiver?
 - o Holger: Stats per subscription should be sufficient. No need for per subscription/pre-receiver stats (which would allow the data model to be simplified a little bit).

- Subscription-ids are currently numeric values with the space split between configured and dynamic subscriptions, but I think that the config model would be cleaner if we used names for the configured subscriptions (and we could reserve a prefix "dyn-" for dynamic subscriptions).
 - o Holger, Rob: Strong preference to names.
 - o Thomas, keep ids, but rename 'purpose' to name. Also give receivers an ids and be keyed this.

D.7.2. Issues related to dynamic subscriptions:

1. In YP Lite, dynamic subscriptions are designed to be closer to configured subscriptions and share more of the data model and lifecycle handling. I.e., the primary differentiator is meant to be how they are instantiated.
 - * James, Rob: More alignment is better.
2. Do we want to change how RPC errors are reported? E.g., change the RPC ok response to indicate whether the subscription was successfully created or not, or included extra error information. Note NETCONF and RESTCONF already define how errors are encoded in XML and JSON (for RESTCONF only).
3. Do we need to allow a dynamic subscription to be modified? If we do, then it would be better to change the establish-subscription RPC to have an optional existing subscription-id rather than define a separate RPC. However, my preference is that the existing subscription is deleted and recreated (or if we allow the client to specify the subscription-id then they could just overwrite the subscription)
 - * Holger: Overwrite is a neat idea, but delete/recreate would also be sufficient.
 - * Rob: Related to this, do we allow the client to optionally name dynamic subscriptions (what is config and dynamic subscription names collide)?
4. For operational data, should we list dynamic receivers in the receiver list so that they are handled the same as configured subscription? Or should the information for them be inlined in the subscriptions container?

- * Rob this requires further thought on exactly how configured/dynamic subscriptions may to underlying transport sessions. I.e., how fixed or flexible is this depending on the transport.

D.8. Issues related to Subscription Lifecycle

1. Use subscription name as the unique identifier for the subscription configuration. Subscription id identifies a subscription session. I.e., if a configured subscription is terminated and re-established then a new subscription id is allocated.
2. Should subscription-started notification include a fingerprint of the schema that is covered by the subscription that would guaranteed to change if the subscription changes?
 - * Tracked via <https://github.com/rgwilton/draft-yp-observability/issues/11>
3. If a subscription references a filter, then should that be included inline in the subscription started notification (as per the RFC 8641 text), or should it indicate that it is a referenced filter?
 - * Thomas: Do we need referenced filters at all? Subscriptions could be simplified if everything was done inline.
 - * gNMI is only done inline.
 - * Juniper also supports filters.
 - * Thomas try to simplify as much as possible, then do we need templating?
4. When a subscription is terminated, should it be MUST NOT send any more notifications after the terminated message, or SHOULD NOT? For a dynamic subscription, should the RPC be synchronous and not reply until it knows that all queues have been drained?
 - * Tracked via <https://github.com/rgwilton/draft-yp-observability/issues/13>
5. Is a publisher allowed to arbitrarily send a sync-on-start resync, e.g., if it detects data loss, or should it always just terminate and reinitialize the subscription?
 - * Holger, terminate & recreate.

- * gNMI, not specified.
 - * Thomas: Keep this as implementation detail.
 - * Sequence-id indicates that a client is dropping message anyway.
 - * Receiver can already monitor and see that there is a problem anyway.
6. If the parameters for a subscription change in any way (e.g., the config changes for a configured subscription, or a referenced filter changes in a dynamic subscription) then do we want to say that the subscription MUST be killed and recreated. I.e., with subscription-terminated/subscription-started notifications? (Section 11)
- * Holger, kill/re-create.
 - * Collector would need to no.
 - * Ebben: Includes addition or deletion of the path.
 - * Benoit: Which parameters are changing, this could impact. It depends. Maybe forcing it down keeps it simple. Would need further definition of what parameters would cause this to be pulled down.
 - * Locally relevant, e.g., modifying transport parameters doesn't force a change.
 - * Benoit: If you are not sure what you are doing you must recreate.
7. Should we have a YANG Action to reset a receiver or a subscription? E.g., discussed in Section 9.3.4.
- * There seems to be consensus that having/keeping such a YANG action is useful if a configured subscription is stuck.
 - * Further discussion is needed to indicate where such an RPC should be in the data model, and what effect it should have:
 - It could be under the subscription list, which would effectively be equivalent to forcing the subscription to terminate and re-initialize across all receivers.

- It could also be under the receiver list, which would effectively be equivalent to forcing all subscriptions using that receiver to terminate and re-initialize.
- *We also need to consider whether this would clear subscription counters or not.*
- *We also consider whether configured subscription MAY/ SHOULD share transport sessions where possible, or whether each subscription uses a separate transport session, or whether this is down to the transport session definition. This decision may impact the design of the data model, and perhaps the transport requirements text.*
- *Rob: I've written up some text for the reset action on a subscription and under the receiver (that applies to all subscriptions that reference the receiver configuration). Unlike the existing reset action, I've removed the return parameter (timestamp of when it took effect), and I think that we should allow it to be processed asynchronously w.r.t the RPC caller. Authors, please check the latest text.*

8. Should we support configurable subscription-level keepalives?

D.9. Issues related to Performance, Reliability & Subscription Monitoring

D.9.1. Issues/questions related to operational data:

1. Should we define some additional operational data to help operators check that the telemetry infrastructure is performing correctly, to get an approximation of the load, etc.

* Rob: probably, but lower priority.

2. Should dynamic subscriptions use the same receivers structure as for configured subscriptions, or should they be inline in the configured subscription?

* Thomas: Two sets of counters, one is at the subscription which is about fetching the data, and the other is on the receiver.

* James: Can think of some uses cases where listing drop counters per subscription may be helpful.

- * Rob: Thinking about it further, it probably needs to be separate, since for some transports, each subscription may open a separate transport session to the receiver rather than trying to mux into a single transport (e.g., TCP).

D.10. Issues related to Conformance and Capabilities

1. Do we advertise that conformance via capabilities and/or YANG features (both for configured and dynamic subscriptions)?
 - * The current doc uses a mixture of both (e.g., features for supporting config vs dynamic subscriptions), capabilities for advertising other capabilities that either cannot be advertised via features, or would arguably be too fine grained.
2. For on-change, should a subscription be rejected (or not brought up) if there are no on-change notifiable nodes? Alternative is to offer implementation flexibility between these two approaches.
 - * Holger: Prefer for the subscription to be rejected.
3. CBOR SID encoding:
 - * In terms of conformance, we have SHOULD for CBOR, but the draft is silent on whether this means with SIDs or not.
 - * Further discussion is required on how to manage SID files.
4. Check of the current approach in the draft of using a separate tree for YP Lite capabilities rather than reusing the YP capabilities (and augmentations) previously defined.
 - * Rob: The draft currently includes a separate tree, which I think is necessary because the supported functionality is different (e.g., dampening), and a implementation may support both YANG Push and YANG Push Lite.
5. Further work and discussion is required for advertising capabilities for filter paths. E.g., listing all of the paths that support on-change could be a very long list. Related, does the draft need to advertise at what points a publisher would decompose a higher subscription into more specific subscriptions.

D.11. Issues related to the YANG Modules

None open.

D.12. Issues related to the Security Considerations (& NACM filtering)

1. Need to consider how NACM applies to YANG Push Lite, which may differ for dynamic vs configured subscription, but generally we want the permissions to be checked when the subscription is created rather than each time a path is accessed.
 - * (James) Take out tight binding to NACM from YANG Push Lite altogether. I.e., decouple YANG Push Lite from what security mechanism is being used.
 - * (Rob) Another choice could be to use NACM as an example rather than the only way.
2. Where should this be in the document (current it in the security considerations section)
3. Do we want to retain the the current text in Section 7 introduction related to terminating a subscription if permissions change?
4. Also note, text was removed from the transport section related to RPC authorization, and which should be moved to an application (rather than transport) layer security mechanism.

D.13. Issues related to the IANA

None open.

D.14. Issues related to the Appendixes

D.14.1. Examples related issues/questions:

1. Not a question, but a note that most of the examples need to be updated to reflect the data models currently in the draft.

D.15. Summary of closed/resolved issues

This appendix is only intended while the authors/WG are working on the document, and should be deleted prior to WG LC.

1. Rename subscription-terminated to subscription-stopped (Change rejected 21 Feb 25, unnecessary renaming.)
2. MUST use envelope, hostname and sequence-number (and event-time) (Decided 21 Feb 25)

3. Don't mandate configured or dynamic subscriptions, allow implementations to implement one or both of them. (Decided 21 Feb 25)
4. Dynamic subscriptions require the encoding to be specified. (Decided 21 Feb 25)
5. DSCP settings are only specified under the receiver (for configured subscriptions) (Decided 21 Feb 25)
6. Config and dynamic subscriptions should be aligned as much as possible.

D.16. Changes

1. Aligned configured and dynamic subscription data models: - Both are configured by name. - Made "purpose" field common and renamed to "description" (limited to 1k max characters, previously unrestricted), i.e., now available for dynamic subscriptions.

Authors' Addresses

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

Holger Keller
Deuetsche Telekom
Email: Holger.Keller@telekom.de

Benoit Claise
Huawei
Email: benoit.claise@huawei.com

Ebben Aries
Juniper
Email: exa@juniper.net

James Cumming
Nokia
Email: james.cumming@nokia.com

Thomas Graf
Swisscom
Email: Thomas.Graf@swisscom.com