Network Management Operations
Internet-Draft
Intended status: Informational
Expires: 5 January 2027

Y. Cui
Tsinghua University
M. Xing
L. Zhang
Zhongguancun Laboratory

4 July 2026

Operational Requirements for Network State Exchange in Agent-Assisted Network Operations
draft-cui-nmop-agent-sketch-com-00

Abstract

This document describes operational requirements for exchanging network state in agent-assisted network operations. In this document, an agent-assisted system is any automation or decision-support system that consumes operational state to support tasks such as anomaly triage, incident correlation, configuration verification, traffic engineering analysis, or attack mitigation support. Such a system may use a large language model (LLM), a rule engine, a statistical model, or conventional software.

The document focuses on operational problems created by high-volume telemetry, cross-domain state sharing, privacy constraints, approximate state summaries, and auditability of state used by automated workflows. It identifies requirements for compact, scoped, mergeable, bounded-error, incrementally synchronized, and auditable network state artifacts. The document complements existing NMOP work on anomaly detection, incident management, and YANG Push/message broker integration. It does not define a new network management protocol, a new agent communication protocol, or a wire format.

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://xmzzyo.github.io/nmop-agent-sketch-com/draft-cui-nmop-agent-sketch-com.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-cui-nmop-agent-sketch-com/.

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

Source for this draft and an issue tracker can be found at https://github.com/xmzzyo/nmop-agent-sketch-com.

Status of This Memo

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

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

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

This Internet-Draft will expire on 5 January 2027.

Table of Contents

1. Introduction

The operational complexity of modern networks has grown substantially. Networks now span multiple autonomous systems (ASes), administrative domains, and technology layers. Network management tasks such as anomaly triage, incident correlation, fault localization, configuration consistency checking, and traffic engineering analysis require timely collection, synthesis, and interpretation of large amounts of network state.

Operators increasingly use automation and decision-support systems to reduce the time required to diagnose and respond to operational events. Some deployments may include agent-assisted components, including LLM-based agents, to help correlate alerts, summarize evidence, generate hypotheses, or prepare recommendations for human review. These components do not remove the need for existing management systems, telemetry pipelines, operator policy, or human accountability.

Agent-assisted operations introduce a practical state exchange problem. An analysis component may need to compare flow behavior across many routers, estimate the number of affected sources, identify configuration drift, or correlate symptoms across domains. Supplying raw telemetry to each component is often impractical because of data volume, privacy constraints, management-plane limits, and the latency of downstream analysis.

This document therefore focuses on requirements for network state artifacts consumed by agent-assisted or automated workflows. These artifacts can be generated downstream of existing telemetry mechanisms, including NETCONF [RFC6241], RESTCONF [RFC8040], YANG

data models [RFC7950], IPFIX [RFC7011], gNMI [GNMI], YANG Push/ message broker pipelines [I-D.ietf-nmop-yang-message-broker-integration], and telemetry message schemas [I-D.ietf-nmop-message-broker-telemetry-message].

1.1. Scope

This document is scoped to the exchange of network state consumed by agent-assisted operational workflows. In this document, "network state" includes telemetry-derived measurements, traffic summaries, topology-related observations, configuration-derived summaries, incident evidence, and other operational facts used to support analysis or recommendations.

The requirements apply to systems in which state may be exchanged between devices, collectors, controllers, domain-level automation components, incident management systems, and agent-assisted analysis components. The requirements are independent of whether the consuming component is implemented using an LLM, a rule engine, a statistical model, or a conventional application.

1.2. Non-Goals

This document does not define a new network management protocol.

This document does not update NETCONF, RESTCONF, IPFIX, CoAP, YANG Push, gNMI, or other existing management and telemetry protocols.

This document does not standardize bindings for any agent communication protocol. Such protocols may be used by particular agent-assisted systems, but the requirements in this document do not depend on them.

This document does not specify autonomous mitigation behavior. High-impact operational actions, such as configuration changes, filtering rules, route policy changes, or rollback operations, remain subject to the authorization, validation, approval, and audit procedures of the deployed network management environment.

This document discusses sketch-based summaries as a candidate technique. It does not require the use of sketches in all deployments and does not define a wire format for sketch exchange.

2. Conventions and Terminology

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

The following terms are used throughout this document:

Agent: A software entity that assists a network management workflow by collecting context, correlating evidence, generating recommendations, or coordinating with other components. An agent may be driven by an LLM, a rule engine, a statistical model, or a combination of methods.

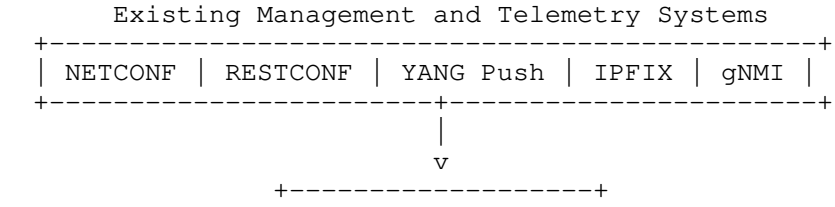
State Artifact: A structured representation of network state exchanged between systems. A state artifact can contain raw state, derived state, or a compact summary, together with metadata describing its scope and provenance.

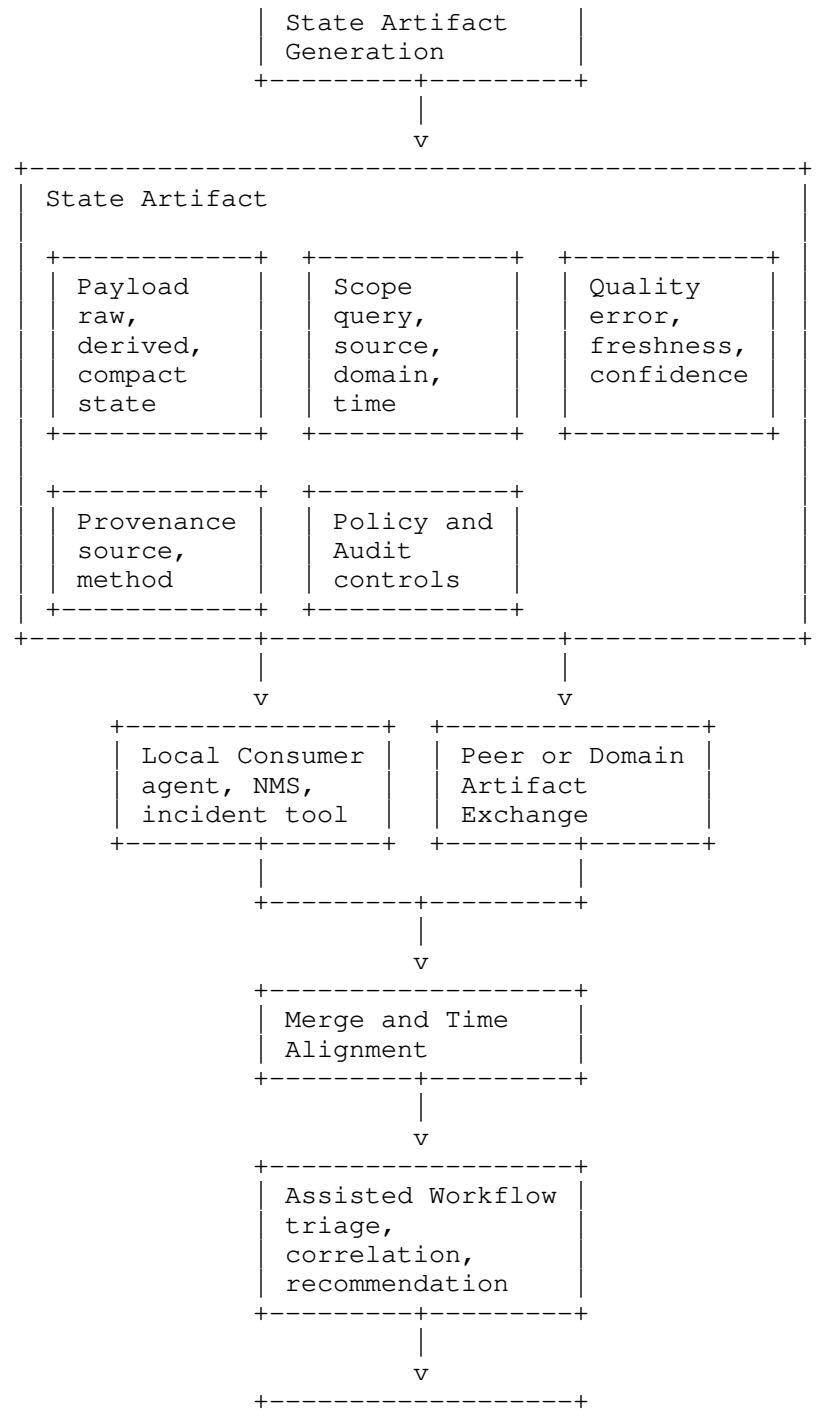
Sketch: A probabilistic data structure that provides a compact, bounded-error summary of a multiset or set of network observations. Examples include Count-Min Sketch, HyperLogLog, DDSketch, MinHash, and Bloom Filter.

Sketch Merge: The operation of combining two Sketch structures of the same type into a single structure whose estimates reflect the union of the underlying observation sets.

3. Conceptual Model and Workflow

The following figure shows a conceptual model for network state exchange in an agent-assisted operational workflow. Existing management and telemetry systems remain the sources of operational data. A generation function produces a state artifact that contains payload, scope, quality, provenance, and policy/audit information. The artifact can then be consumed locally, exchanged with another domain, merged with other artifacts, or referenced as evidence by an assisted workflow.




State Artifact Generation
State Artifact
Payload raw, derived, compact state
Scope query, source, domain, time
Quality error, freshness, confidence
Provenance source, method
Policy and Audit controls
Local Consumer agent, NMS, incident tool
Peer or Domain Artifact Exchange
Merge and Time Alignment
Assisted Workflow triage, correlation, recommendation

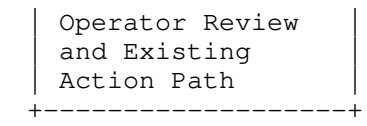


Figure 1: Conceptual state artifact model and workflow

The requirements in this document describe the expected properties of the state artifact and of the workflow steps that generate, exchange, merge, and consume it. The figure is illustrative and does not define a protocol architecture or a required deployment topology.

4. Problem Statement

Agent-assisted operational workflows may require broad network context across many devices and time windows. For example, incident triage may need recent interface counters, flow records, routing changes, topology information, device health indicators, and configuration differences. Sending raw telemetry to each analysis component can create excessive storage, transport, and processing overhead.

Operational workflows often need answers to specific questions rather than full raw records. Examples include:

* Which source prefixes are likely to be heavy hitters during an attack?

* How many distinct sources are observed across a domain?

* Which links show latency distributions that differ from a baseline?

* Which devices have configuration sets that are inconsistent with their peers?

* Which flows are likely affected by a reported incident?

Raw telemetry can answer these questions, but it may be unnecessarily expensive to exchange. Conversely, a summary that is too lossy, lacks error metadata, or cannot be audited can mislead an automated workflow. The challenge is to represent state at the right granularity for the operational question.

Some incidents cross administrative boundaries. Examples include distributed denial-of-service attacks, inter-domain reachability failures, and multi-provider service incidents. In such cases, operators may need to share selected state with peers or with a

coordinating system. However, raw flow records, customer identifiers, topology details, and configuration fragments may be sensitive or subject to policy restrictions.

Compact or approximate state summaries can reduce data movement, but they introduce uncertainty. If an agent-assisted workflow consumes approximate state without understanding error bounds, time coverage, source scope, or freshness, it may generate incorrect conclusions or overconfident recommendations.

Existing telemetry and management mechanisms provide important capabilities, including schema-driven configuration and state access, streaming telemetry, flow export, and message-broker integration. However, deployments that introduce agent-assisted analysis still need guidance on what properties exchanged state artifacts should have so that they are compact, mergeable, bounded in error, privacy-aware, and auditable.

5. Requirements

This section defines initial operational requirements for network state exchange in agent-assisted network operations. The list is intended as a starting point for discussion.

5.1. Compact State Representation (REQ-1)

Solutions SHOULD support network state representations that are substantially smaller than the raw telemetry, logs, flow records, or configuration data from which they are derived, when the operational query does not require full-fidelity raw data.

The compact representation MUST preserve enough information to answer the intended operational query within the accuracy, freshness, and confidence requirements of the workflow.

5.2. Query and Scope Metadata (REQ-2)

An exchanged state artifact MUST identify the operational query or query class that it is intended to support.

An exchanged state artifact MUST identify its scope, including the source device set or domain, observation time interval, collection method, sampling policy if any, and relevant aggregation parameters.

An exchanged state artifact SHOULD include provenance metadata sufficient for an operator or an incident management system to trace the artifact back to the telemetry source, collector, or generation process. This is aligned with the provenance needs described by telemetry message work in NMOP [I-D.ietf-nmop-message-broker-telemetry-message].

5.3. Bounded and Reportable Error (REQ-3)

If a state artifact is approximate, the artifact MUST include the parameters needed to interpret its error behavior.

For probabilistic summaries, the artifact MUST report the applicable error parameters, such as relative error, false-positive probability, confidence level, or other algorithm-specific bounds.

Consumers of approximate state SHOULD treat the error metadata as part of the input to their reasoning and SHOULD expose uncertainty in any generated recommendation, incident report, or operator-facing explanation.

5.4. Mergeability Across Devices and Domains (REQ-4)

Solutions SHOULD support merging of state artifacts from multiple devices, collectors, or administrative domains when the operational query requires aggregate visibility.

A merge operation MUST preserve or update the scope metadata of the resulting artifact so that the combined device set, domain set, and time interval are explicit.

When merging approximate artifacts, the resulting artifact MUST include updated error metadata or indicate that the error behavior is unknown.

5.5. Freshness and Time Alignment (REQ-5)

An exchanged state artifact MUST include timestamp information sufficient to determine its freshness.

When a workflow combines artifacts from multiple sources, the system SHOULD make the observation windows visible to the consumer so that time skew and stale inputs can be detected.

5.6. Incremental Synchronization (REQ-6)

Solutions SHOULD support incremental updates when only a subset of the represented state changes between synchronization points.

Solutions MUST provide a way to detect stale, missing, or inconsistent updates when incremental synchronization is used.

Solutions MUST support full resynchronization when incremental updates are incomplete or when the receiver cannot reconstruct the current state.

5.7. Privacy-Preserving Exchange (REQ-7)

Solutions SHOULD support cross-domain state exchange without requiring disclosure of raw flow records, customer identifiers, full topology details, or other sensitive operational data when aggregate state is sufficient.

Solutions MUST make clear whether a state artifact may still leak sensitive information through keys, labels, repeated queries, low-cardinality sets, or correlations with external data.

Operators SHOULD be able to apply policy controls to determine which state artifacts may be shared, with whom, and at what granularity.

5.8. Auditability and Reproducibility (REQ-8)

State artifacts used by agent-assisted workflows SHOULD be logged or referenced in a way that allows later audit of the evidence used by the workflow.

The audit record SHOULD include artifact identifiers, generation parameters, source scope, time interval, software or model version where applicable, and consumer identity.

When an operator-facing recommendation is generated from approximate state, the recommendation SHOULD identify the input artifacts and their uncertainty metadata.

5.9. Interoperability with Existing Management Systems (REQ-9)

Solutions SHOULD integrate with existing network management and telemetry systems rather than requiring a parallel data collection infrastructure.

Solutions SHOULD be able to consume state derived from existing mechanisms such as NETCONF [RFC6241], RESTCONF [RFC8040], YANG-modeled data [RFC7950], IPFIX [RFC7011], message brokers, time-series databases, and controller APIs.

Compact state artifacts generated downstream of YANG Push/message broker pipelines SHOULD preserve useful source and schema metadata from those pipelines [I-D.ietf-nmop-yang-message-broker-integration].

5.10. Separation from Operational Action (REQ-10)

State exchange mechanisms MUST NOT by themselves imply authorization to perform operational actions.

High-impact actions that are influenced by exchanged state, such as filtering, routing changes, configuration updates, or rollback operations, MUST remain subject to the authorization, validation, approval, and audit procedures of the deployment.

Approximate state SHOULD NOT be the sole basis for unattended high-impact action unless the operator has explicitly defined the applicable policy, risk threshold, validation process, and rollback procedure.

6. Relationship to NMOP Work

This document is intended to complement existing NMOP work, rather than replace it.

The network anomaly architecture [I-D.ietf-nmop-network-anomaly-architecture], anomaly lifecycle [I-D.ietf-nmop-network-anomaly-lifecycle], and anomaly semantics [I-D.ietf-nmop-network-anomaly-semantics] describe how operational evidence can be collected, annotated, validated, and used in anomaly detection workflows. The requirements in this document focus on the properties of state artifacts that may feed such workflows.

The network incident YANG model [I-D.ietf-nmop-network-incident-yang] provides a structure for incident management. Compact state artifacts can be referenced as incident evidence or diagnostic inputs.

The YANG Push/message broker integration work [I-D.ietf-nmop-yang-message-broker-integration] and telemetry message model [I-D.ietf-nmop-message-broker-telemetry-message] address telemetry transport, schema, and provenance. Compact state artifacts can be generated downstream of such pipelines and should preserve relevant provenance and scope metadata.

7. Example Candidate: Sketch-Based State Summaries

Sketch structures are one candidate representation for compact state artifacts exchanged by agent-assisted workflows. Rather than transmitting raw flow records, routing tables, or interface statistics for every query, a deployment can exchange Sketch summaries that answer specific questions about network state with bounded or measurable error.

A Sketch is not a detection tool or anomaly detector. It is a candidate summary artifact that may be consumed by operational systems. Its usefulness depends on the operational query, selected parameters, acceptable error bounds, and audit requirements.

The appropriate Sketch type depends on the nature of the network state being represented and the queries agents need to answer:

Operational Task	Query Type	Candidate Summary	Key Property Used
Flow rate analysis	"What is the traffic rate from prefix X?"	Count-Min Sketch (CMS)	Frequency estimation with epsilon-delta bounds
Source diversity analysis	"How many unique source IPs are there?"	HyperLogLog (HLL)	Cardinality estimation, cross-domain mergeable
Latency / jitter analysis	"What is the p99 latency on path P?"	DDSketch	Quantile estimation with relative error bounds
Configuration consistency	"Is device A's config consistent with peers?"	MinHash	Set similarity estimation (Jaccard index)
Affected flow marking	"Is flow F affected by fault X?"	Bloom Filter	Set membership with configurable false positive rate

Table 1

If Sketches are used, their artifacts need to carry scope, freshness, provenance, and error metadata as described in the requirements above.

8. Initial Use Cases

This section lists initial use cases. More detailed operator use cases are expected in future revisions.

DDoS evidence exchange: A domain can publish a compact heavy-hitter or cardinality summary as evidence for an incident workflow. A peer or coordinating system can use the artifact to assess whether an attack appears distributed without requiring raw flow records. Mitigation, if any, is performed through existing mechanisms such as FlowSpec [RFC8955] or local filtering procedures.

Multi-domain fault localization: Domains can exchange latency or loss distribution summaries for aligned time windows. A coordinating workflow can identify where behavior deviates from baseline and request additional evidence from the relevant domain.

Configuration consistency verification: A collector can publish compact configuration-set similarity artifacts. An audit workflow can identify devices that appear inconsistent and hand them to existing configuration management systems for operator review.

Traffic engineering analysis: A traffic engineering workflow can consume summarized traffic matrix and latency artifacts to prepare recommendations. Any approved routing or policy change is applied through existing operational procedures.

9. Security Considerations

State artifacts exchanged over untrusted networks need authentication, integrity protection, and confidentiality appropriate to the sensitivity of the represented state.

Credential management should bind a consuming component to an operational role, administrative domain, and permitted state scope.

Sketch structures or other compact state artifacts could be tampered with to influence agent-assisted analysis. Transport security can prevent eavesdropping and impersonation, but deployments should also consider artifact-level integrity protection where artifacts are stored, forwarded, or consumed asynchronously.

An adversary with write access to a summary generation function could manipulate summaries to cause incorrect analysis or recommendations. Defenses include using keyed hash functions such as SipHash [SIPHASH] for Sketch index computation, cross-validating estimates from multiple independent sources, and monitoring for statistically anomalous summary patterns.

State generation and exchange functions can be targets for denial-of-service attacks. Implementations should enforce rate limits, quotas, back pressure, and admission control for artifact generation and retrieval.

Approximate state summaries can also leak information through repeated queries, low-cardinality sets, or correlation with external data. Sharing policies need to account for such leakage risks.

10. IANA Considerations

This document has no IANA actions.

11. References

11.1. Normative References

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

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

11.2. Informative References

[CORMODE] Cormode, G. and S. Muthukrishnan, "An Improved Data Stream Summary The Count-Min Sketch and its Applications", n.d..

[GNMI] "gRPC Network Management Interface (gNMI)", 2018, <https://datatracker.ietf.org/doc/html/draft-openconfig-rtgwg-gnmi-spec>.

[I-D.ietf-nmop-message-broker-telemetry-message]
Elhassany, A., Graf, T., and P. Lucente, "Extensible YANG Model for Network Telemetry Messages", Work in Progress, Internet-Draft, draft-ietf-nmop-message-broker-telemetry-message-04, 18 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-message-broker-telemetry-message-04>.

[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-07, 18 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-network-anomaly-architecture-07>.

[I-D.ietf-nmop-network-anomaly-lifecycle]
Riccobene, V., Graf, T., Du, W., and A. H. Feng, "An Experiment: Network Anomaly Detection Lifecycle", Work in Progress, Internet-Draft, draft-ietf-nmop-network-anomaly-lifecycle-05, 12 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-network-anomaly-lifecycle-05>.

[I-D.ietf-nmop-network-anomaly-semantics]
Graf, T., Du, W., Feng, A. H., and V. Riccobene, "Semantic Metadata Annotation for Network Anomaly Detection", Work in Progress, Internet-Draft, draft-ietf-nmop-network-anomaly-semantics-05, 19 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-network-anomaly-semantics-05>.

[I-D.ietf-nmop-network-incident-yang]
Hu, T., Contreras, L. M., Wu, Q., Davis, N., and C. Feng, "A YANG Data Model for Network Incident Management", Work in Progress, Internet-Draft, draft-ietf-nmop-network-incident-yang-09, 28 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-network-incident-yang-09>.

[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-13, 2 July 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-yang-message-broker-integration-13>.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", n.d..

[RFC7011] Claise, B., Trammell, B., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", n.d..

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", n.d..

[RFC8955] Loibl, C., Hares, S., Raszuk, R., McPherson, D., and M. Bacher, "Dissemination of Flow Specification Rules", n.d..

[SIPHASH] Aumasson, J.-P. and D. J. Bernstein, "SipHash A Fast Short-Input PRF", n.d..

Acknowledgments

TODO acknowledge.

Authors' Addresses

Yong Cui
Tsinghua University
Beijing, 100084
China
Email: cuiyong@tsinghua.edu.cn
URI: http://www.cuiyong.net/

Mingzhe Xing
Zhongguancun Laboratory
Beijing, 100094
China
Email: xingmz@zgclab.edu.cn

Lei Zhang
Zhongguancun Laboratory
Beijing, 100094
China
Email: zhanglei@zgclab.edu.cn

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 1 January 2027

M. Jethanandani
Kloud Services
K. Watsen
Watsen Networks

30 June 2026

An HTTPS-based Transport for YANG Notifications
draft-ietf-netconf-https-notif-16

Abstract

This document defines a protocol for sending asynchronous event notifications similar to notifications defined in RFC 5277, but over HTTPS. YANG modules for configuring publishers are also defined. Examples are provided illustrating how to configure various publishers.

This document requires that the publisher is a "server" (e.g., a NETCONF or RESTCONF server), but does not assume that the receiver is a server.

Table of Contents

1. Introduction

This document defines a protocol for sending asynchronous event notifications similar to notifications defined in NETCONF Event Notifications [RFC5277], but over HTTPS. Using HTTP Semantics [RFC9110], which maximizes transport-level interoperability, while allowing for a variety of encoding options. The protocol supports HTTP/1.1: Message Syntax and Routing [RFC9112] and, HTTP/2 [RFC9113]. While the payload does not change between these versions of HTTP and HTTP/3 [RFC9114], the underlying transport does. Since there is no support for configuring the minimum required parameters to enable notifications over HTTP/3 [RFC9114], support for it is considered out of scope of this document at this time.

This document defines support for JSON and XML content, using the media types defined in RESTCONF [RFC8040]; future efforts may define support for other encodings (e.g., CBOR). This document also defines optional support for sending legacy notifications as defined in NETCONF Event Notifications [RFC5277] using "application/xml" as the media type. This document requires that the publisher is a network management server (e.g., a NETCONF or RESTCONF server), but does not require that the receiver also be a network management server. The receiver is, however, required to be an HTTPS server capable of receiving the notifications.

This document also defines two YANG 1.1 [RFC7950] modules that extend the data model defined in Subscription to YANG Notifications [RFC8639], enabling the configuration of HTTPS-based receivers.

An example module illustrating the configuration of a publisher not using the data model defined in RFC 8639 is also provided.

Configured subscriptions enable a server (e.g., a NETCONF or RESTCONF server), acting as a publisher of notifications, to proactively push notifications to external receivers without the receivers needing to first connect to the server, as is the case with dynamic subscriptions.

1.1. Applicability Statement

While the YANG modules have been defined as an augmentation of Subscription to YANG Notifications [RFC8639], the notification method defined in this document MAY be used outside of Subscription to YANG Notifications [RFC8639] by using some of the definitions from this module along with the grouping defined in Groupings for HTTP Clients and Servers [I-D.ietf-netconf-http-client-server]. For an example on how that can be done, see Section A.2.

1.2. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this section before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

RFC YYYY, where YYYY is the number assigned to [I-D.ietf-netconf-http-client-server].

2026-06-30 with the actual date of the publication of this document.

1.3. Abbreviations

Acronym	Expansion
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SSE	Server-Sent Events
TCP	Transmission Control Protocol
TLS	Transport Layer Security

Table 1

1.4. Terminology

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

1.4.1. Terms Imported from other RFCs

The following terms are defined in Subscription to YANG Notifications [RFC8639].

* Publisher

* Receiver

* Subscribed Notifications

The following term is defined in RESTCONF Protocol [RFC8040].

* target resource

1.5. Tree Diagram

The tree diagram for the YANG modules defined in this document use annotations defined in YANG Tree Diagrams. [RFC8340].

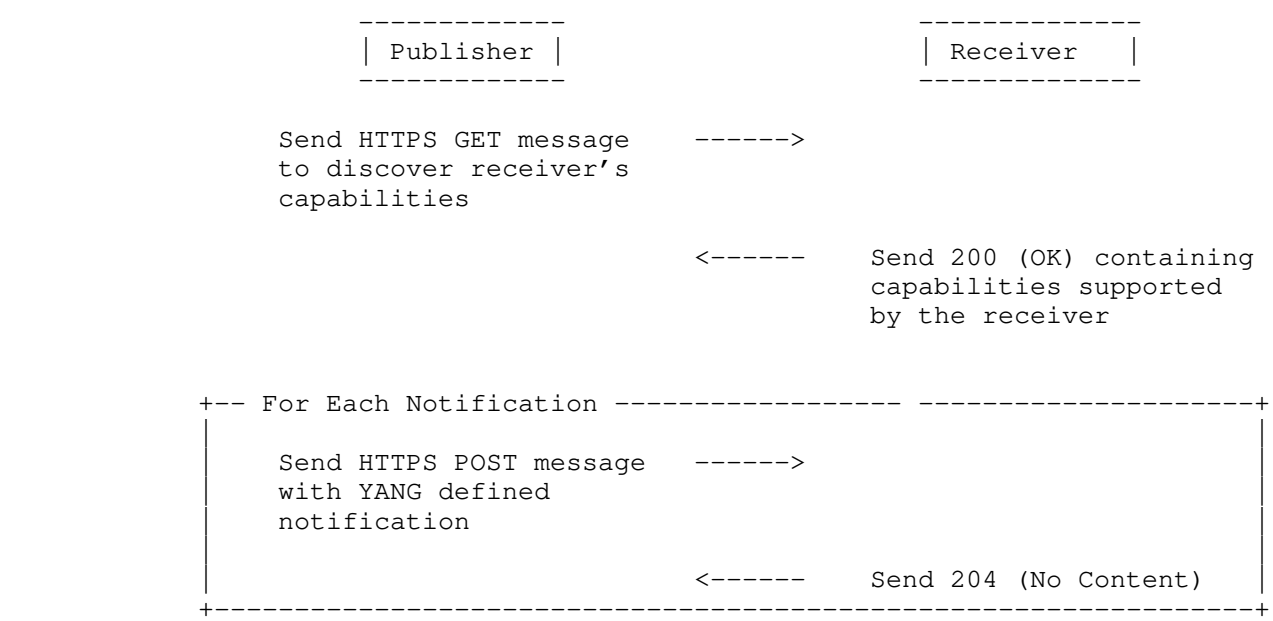
2. Overview of Publisher to Receiver Interaction

The protocol consists of two HTTP-based target resources presented by the receiver. These two resources share a common prefix that the publisher learns from a request it issues, as defined in section 3.2. If the data model in section 6.2 is used, this common prefix is defined by the "path" leaf in the "http-client-parameters" container.

* "capabilities": A target resource enabling the publisher to discover what optional capabilities a receiver supports. Publishers SHOULD query this target before sending any notifications or if ever an error occurs.

* "relay-notification": A target resource enabling the publisher to send one or more notification to a receiver. This document defines support for sending only one notification per message; a future effort MAY extend the protocol to send multiple notifications per message.

The protocol is illustrated in the diagram below:



Note that, for RFC 8639 configured subscriptions, the very first notification must be the "subscription-started" notification.

3. Discovering a Receiver's Capabilities

3.1. Applicability

For publishers using Subscription to YANG Notifications [RFC8639], dynamic discovery of a receiver's supported encoding is necessary only when the "/subscriptions/subscription/encoding" leaf is not configured, per the "encoding" leaf's description statement in the "ietf-subscribed-notification" module.

If the "encoding" leaf is not configured, and the publisher wants to send a notification in a particular format, without going through the setup operation of learning the receiver capabilities, it can do so, but has to be prepared for the case when it receives an error response, because the receiver does not support the format sent by the publisher.

3.2. Request

To learn the capabilities of a receiver, a publisher can issue an HTTPS GET request to the "capabilities" resource (see Section 2) on the receiver with the "Accept" header set to "application/yang-data+xml" and/or "application/yang-data+json", as defined in RESTCONF [RFC8040].

3.3. Response

The receiver responds with a "200 (OK)" message, having the "Content-Type" header set to either "application/yang-data+xml" or "application/yang-data+json" (whichever was selected), and containing in the response body a list of the receiver's capabilities encoded in the selected format.

The response body MUST conform to the "receiver-capabilities" structure defined in the "ietf-https-notif-transport" YANG module (see YANG Data Structure Extensions [RFC8791]), which is shown below:

```
structure receiver-capabilities {
  description
    "A structure defining the response body for the capabilities
     GET request.";
  leaf-list receiver-capability {
    type inet:uri;
    description
      "A capability supported by the receiver.  A partial list of
       capabilities is defined in the 'Capabilities for HTTPS
       Notification Receivers' registry (see RFC XXXX). Additional
       custom capabilities MAY be defined.";
  }
}
```

As it is possible that the receiver may return custom capability URIs, the publisher MUST ignore any capabilities that it does not recognize.

3.4. Example

The publisher can send the following request to learn the receiver capabilities. In this example, the "Accept" states that the publisher wants to receive the capabilities response in XML but, if not supported, then in JSON.

```
GET /some/path/capabilities HTTP/1.1
Host: example.com
Accept: application/yang-data+xml, application/yang-data+json;q=0.5
```

If the receiver is able to reply using "application/yang-data+xml", and assuming it is able to receive JSON and XML encoded notifications, and it is able to process the RFC 8639 state machine, the response might look like this:

```
HTTP/1.1 200 OK
Date: Wed, 26 Feb 2020 20:33:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+xml

<receiver-capabilities
    xmlns="urn:ietf:params:xml:ns:yang:ietf-https-notif-transport">
  <receiver-capability>
    urn:ietf:params:yang-notif:https-capability:encoding:json
  </receiver-capability>
  <receiver-capability>
    urn:ietf:params:yang-notif:https-capability:encoding:xml
  </receiver-capability>
  <receiver-capability>
    urn:ietf:params:yang-notif:https-capability:sub-notif
  </receiver-capability>
</receiver-capabilities>
```

If the receiver is unable to reply using "application/yang-data+xml", the response might look like this:

```
HTTP/1.1 200 OK
Date: Wed, 26 Feb 2020 20:33:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json
Content-Length: nnn

{
  "ietf-https-notif-transport:receiver-capabilities": {
    "receiver-capability": [
      "urn:ietf:params:yang-notif:https-capability:encoding:json",
      "urn:ietf:params:yang-notif:https-capability:encoding:xml",
      "urn:ietf:params:yang-notif:https-capability:sub-notif"
    ]
  }
}
```

4. Sending Event Notifications

4.1. Request

The publisher sends an HTTP POST request to the "relay-notification" resource (see Section 2) on the receiver with the "Content-Type" header set to either "application/yang-data+json" or "application/yang-data+xml" and a body containing the notification encoded using the specified format.

XML-encoded notifications are encoded using the format defined by NETCONF Event Notifications [RFC5277] for XML.

JSON-encoded notifications are encoded the same as specified in Section 6.4 in RESTCONF [RFC8040] with the following deviations:

* The notifications do not contain the "data:" prefix used by Server-Sent Events (SSE).

* Instead of saying that, for JSON-encoding purposes, the module name for the "notification" element is "ietf-restconf", the module name will instead be "ietf-https-notif".

4.2. Response

The response on success SHOULD be from the 2XX class of codes. In case of corrupted or malformed event, the response SHOULD be an appropriate HTTP error response.

4.3. Example

An XML-encoded notification might be sent as follows:

```
POST /some/path/relay-notification HTTP/1.1
Host: example.com
Content-Type: application/yang-data+xml

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2019-03-22T12:35:00Z</eventTime>
  <event xmlns="https://example.com/example-mod">
    <event-class>fault</event-class>
    <reporting-entity>
      <card>Ethernet0</card>
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

A JSON-encoded notification might be sent as follows:

```
POST /some/path/relay-notification HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json

{
  "ietf-https-notif:notification": {
    "eventTime": "2013-12-21T00:01:00Z",
    "example-mod:event" : {
      "event-class" : "fault",
      "reporting-entity" : { "card" : "Ethernet0" },
      "severity" : "major"
    }
  }
}
```

And, in either case, the response on success might be as follows:

```
HTTP/1.1 204 No Content
Date: Wed, 26 Feb 2020 20:33:30 GMT
Server: example-server
```

5. Support for Legacy Notifications (RFC 5277)

This document primarily defines a transport for YANG notifications as defined in Subscription to YANG Notifications [RFC8639]. However, implementations MAY also use this HTTPS-based transport to send legacy event notifications as defined in NETCONF Event Notifications [RFC5277]. This section defines how such legacy notifications are sent and how a receiver advertises support for them.

5.1. Request

When sending legacy RFC 5277 notifications, the publisher sends an HTTP POST request to the "relay-notification" resource (see Section 2) with the "Content-Type" header set to "application/xml". The body MUST contain the notification encoded as defined in NETCONF Event Notifications [RFC5277].

5.2. Capabilities

A receiver that supports legacy RFC 5277 notifications MUST advertise this capability by including the following URI in its capabilities response (see Section 2):

urn:ietf:params:yang-notif:https-capability:rfc5277-notif

A publisher MUST NOT send RFC 5277 legacy notifications unless it has confirmed that the receiver supports the "rfc5277-notif" capability URI. The publisher MUST confirm this either by querying the receiver's capabilities as described in Section 2, or through prior knowledge (e.g., static configuration).

A YANG-aware publisher SHOULD issue the capabilities request with an "Accept" header set to "application/yang-data+xml" and/or "application/yang-data+json", as described in Section 2. A legacy publisher that only understands "application/xml" MAY instead set the "Accept" header to "application/xml". In that case, the receiver SHOULD respond with "Content-Type: application/xml" and the same "receiver-capabilities" XML structure, allowing the legacy publisher to parse the capability URIs without requiring knowledge of YANG media types.

5.3. Example

A legacy publisher that only understands "application/xml" can discover receiver capabilities as follows:

```
GET /some/path/capabilities HTTP/1.1
Host: example.com
Accept: application/xml
```

The receiver, recognizing that the publisher sent "Accept: application/xml", responds with the same "receiver-capabilities" structure but using "Content-Type: application/xml":

```
HTTP/1.1 200 OK
Date: Wed, 26 Feb 2020 20:33:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/xml

<receiver-capabilities
    xmlns="urn:ietf:params:xml:ns:yang:ietf-https-notif-transport">
  <receiver-capability>
    urn:ietf:params:yang-notif:https-capability:encoding:json
  </receiver-capability>
  <receiver-capability>
    urn:ietf:params:yang-notif:https-capability:encoding:xml
  </receiver-capability>
  <receiver-capability>
    urn:ietf:params:yang-notif:https-capability:rfc5277-notif
  </receiver-capability>
</receiver-capabilities>
```

Having confirmed the "rfc5277-notif" capability, the legacy publisher then sends a notification as follows:

```
POST /some/path/relay-notification HTTP/1.1
Host: example.com
Content-Type: application/xml

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2019-03-22T12:35:00Z</eventTime>
  <event xmlns="https://example.com/example-mod">
    <event-class>fault</event-class>
    <reporting-entity>
      <card>Ethernet0</card>
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

And the response on success might be as follows:

```
HTTP/1.1 204 No Content
Date: Wed, 26 Feb 2020 20:33:30 GMT
Server: example-server
```

6. The "ietf-subscribed-notif-receivers" Module

6.1. Data Model Overview

This YANG module augments the "ietf-subscribed-notifications" module to define a choice of transport types that other modules such as the "ietf-https-notif-transport" module can use to define a transport specific receiver.

module: ietf-subscribed-notif-receivers

```
  augment /sn:subscriptions:
    +--rw receiver-instances
       +--rw receiver-instance* [name]
          +--rw name    string
          +--rw (transport-type)
  augment /sn:subscriptions/sn:subscription/sn:receivers/sn:receiver:
    +--rw receiver-instance-ref?   leafref
```

6.2. YANG Module

The YANG module imports Subscription to YANG Notifications [RFC8639].

```
<CODE BEGINS> file "ietf-subscribed-notif-receivers@2026-06-30.yang"
module ietf-subscribed-notif-receivers {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notif-receivers";
  prefix "snr";

  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }

  organization
    "IETF NETCONF Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/netconf>
     WG List:  <netconf@ietf.org>

     Authors: Mahesh Jethanandani (mjethanandani at gmail dot com)
              Kent Watsen (kent plus ietf at watsen dot net)";

  description
    "This YANG module is implemented by Publishers implementing
     the 'ietf-subscribed-notifications' module defined in RFC 8639.

     While this module is defined in RFC XXXX, which primarily
     defines an HTTPS-based transport for notifications, this module
     is not HTTP-specific.  It is a generic extension that can be
     used by any 'notif' transport.

     This module defines two 'augment' statements.  One statement
     augments a 'container' statement called 'receiver-instances'
     into the top-level 'subscriptions' container.  The other
     statement, called 'receiver-instance-ref', augments a 'leaf'
     statement into each 'receiver' that references one of the
     afore mentioned receiver instances.  This indirection enables
     multiple configured subscriptions to send notifications to
     the same receiver instance.

     Copyright (c) 2024 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.
     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the Revised BSD
     License set forth in Section 4.c of the IETF Trust's Legal
     Provisions Relating to IETF Documents
```

```
        (http://trustee.ietf.org/license-info).

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

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
        'MAY', and 'OPTIONAL' in this document are to be interpreted as
        described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

    revision "2026-06-30" {
      description
        "Initial Version.";
      reference
        "RFC XXXX: An HTTPS-based Transport for YANG Notifications.";
    }

    augment "/sn:subscriptions" {
      container receiver-instances {
        description
          "A container for all instances of receivers.";

        list receiver-instance {
          key "name";

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

          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.";
        }
      }
      description
        "Augment the subscriptions container to define the
         transport type.";
    }
```

```
    augment
      "/sn:subscriptions/sn:subscription/sn:receivers/sn:receiver" {
      leaf receiver-instance-ref {
        type leafref {
          path "/sn:subscriptions/snr:receiver-instances/" +
               "snr:receiver-instance/snr:name";
        }
        description
          "Reference to a receiver instance.";
      }
      description
        "Augment the subscriptions container to define an optional
         reference to a receiver instance.";
    }
  }
  <CODE ENDS>
```

7. The "ietf-https-notif-transport" Module

7.1. Data Model Overview

This YANG module is a definition of a set of receivers that are interested in the notifications published by the publisher. The module contains the TCP, TLS and HTTPS parameters that are needed to communicate with the receiver. The module augments the "ietf-subscribed-notif-receivers" module to define a transport specific receiver.

As mentioned earlier, it uses a POST method to deliver the notification. The "http-receiver/tls/http-client-parameters/path" leaf defines the path for the resource on the receiver, as defined by "path-absolute" in URI Generic Syntax [RFC3986]. The user-id used by Network Configuration Access Control Model [RFC8341], is that of the receiver and is derived from the certificate presented by the receiver as part of "receiver-identity".

An abridged tree diagram representing the module is shown below.

```
module: ietf-https-notif-transport

  augment /sn:subscriptions/snr:receiver-instances
            /snr:receiver-instance/snr:transport-type:
    +--:(https)
       +--rw https-receiver
          +--rw capabilities-endpoint
          |  +--rw uri
          |  |  +--rw scheme       string
          |  |  +--rw authority!
```

```
|  |  |  +--rw userinfo?   string
|  |  |  +--rw host        inet:host
|  |  |  +--rw port?       inet:port-number
|  |  +--rw path?        string
|  |  +--rw query?       string
|  |  +--rw fragment?    string
|  +--rw protocol-versions!
|  |  +--rw protocol-version*   httpv:http-protocol-version
|  +--rw tls-client-parameters!
|  |  +--rw client-identity!
|  |  |  +--rw (auth-type)
|  |  |     +--:(certificate) {client-ident-x509-cert}?
|  |  |     |     ...
|  |  |     +--:(raw-public-key)
|  |  |     |        {client-ident-raw-public-key}?
|  |  |     |     ...
|  |  |     +--:(tls12-psk) {client-ident-tls12-psk}?
|  |  |     |     ...
|  |  |     +--:(tls13-epsk) {client-ident-tls13-epsk}?
|  |  |           ...
|  |  +--rw server-authentication
|  |  |  +--rw ca-certs! {server-auth-x509-cert}?
|  |  |  |  +--rw (inline-or-truststore)
|  |  |  |        ...
|  |  |  +--rw ee-certs! {server-auth-x509-cert}?
|  |  |  |  +--rw (inline-or-truststore)
|  |  |  |        ...
|  |  |  +--rw raw-public-keys!
|  |  |  |       {server-auth-raw-public-key}?
|  |  |  |  +--rw (inline-or-truststore)
|  |  |  |        ...
|  |  |  +--rw tls12-psks?        empty
|  |  |  |       {server-auth-tls12-psk}?
|  |  |  +--rw tls13-epsks?       empty
|  |  |           {server-auth-tls13-epsk}?
|  |  +--rw hello-params {tlscmn:hello-params}?
|  |  |  +--rw tls-versions
|  |  |  |  +--rw min?   identityref
|  |  |  |  +--rw max?   identityref
|  |  |  +--rw cipher-suites
|  |  |     +--rw cipher-suite*
|  |  |              tlscsa:tls-cipher-suite-algorithm
|  |  +--rw keepalives {tls-client-keepalives}?
|  |     +--rw peer-allowed-to-send?   empty
|  |     +--rw test-peer-aliveness!
|  |        +--rw max-wait?       uint16
|  |        +--rw max-attempts?   uint8
|  +--rw proxy-connect! {proxy-connect}?
```

```
|  |  +--rw protocol    enumeration
|  |  +--rw host        inet:host
|  |  +--rw port        uint16
|  +--ro supported-versions {version-discovery}?
|     +--ro supported-version*
|              httpv:http-protocol-version
+--rw relay-notification-endpoint
|  +--rw uri
|  |  +--rw scheme       string
|  |  +--rw authority!
|  |  |  +--rw userinfo?   string
|  |  |  +--rw host        inet:host
|  |  |  +--rw port?       inet:port-number
|  |  +--rw path?        string
|  |  +--rw query?       string
|  |  +--rw fragment?    string
|  +--rw protocol-versions!
|  |  +--rw protocol-version*   httpv:http-protocol-version
|  +--rw tls-client-parameters!
|  |  +--rw client-identity!
|  |  |  +--rw (auth-type)
|  |  |     +--:(certificate) {client-ident-x509-cert}?
|  |  |     |     ...
|  |  |     +--:(raw-public-key)
|  |  |     |        {client-ident-raw-public-key}?
|  |  |     |     ...
|  |  |     +--:(tls12-psk) {client-ident-tls12-psk}?
|  |  |     |     ...
|  |  |     +--:(tls13-epsk) {client-ident-tls13-epsk}?
|  |  |           ...
|  |  +--rw server-authentication
|  |  |  +--rw ca-certs! {server-auth-x509-cert}?
|  |  |  |  +--rw (inline-or-truststore)
|  |  |  |        ...
|  |  |  +--rw ee-certs! {server-auth-x509-cert}?
|  |  |  |  +--rw (inline-or-truststore)
|  |  |  |        ...
|  |  |  +--rw raw-public-keys!
|  |  |  |       {server-auth-raw-public-key}?
|  |  |  |  +--rw (inline-or-truststore)
|  |  |  |        ...
|  |  |  +--rw tls12-psks?        empty
|  |  |  |       {server-auth-tls12-psk}?
|  |  |  +--rw tls13-epsks?       empty
|  |  |          {server-auth-tls13-epsk}?
|  |  +--rw hello-params {tlscmn:hello-params}?
|  |  |  +--rw tls-versions
|  |  |  |  +--rw min?   identityref
```

```
              |  |  |  |  +--rw max?   identityref
              |  |  |  +--rw cipher-suites
              |  |  |     +--rw cipher-suite*
              |  |  |              tlscsa:tls-cipher-suite-algorithm
              |  |  +--rw keepalives {tls-client-keepalives}?
              |  |     +--rw peer-allowed-to-send?   empty
              |  |     +--rw test-peer-aliveness!
              |  |        +--rw max-wait?       uint16
              |  |        +--rw max-attempts?   uint8
              |  +--rw proxy-connect! {proxy-connect}?
              |  |  +--rw protocol    enumeration
              |  |  +--rw host        inet:host
              |  |  +--rw port        uint16
              |  +--ro supported-versions {version-discovery}?
              |     +--ro supported-version*
              |              httpv:http-protocol-version
              +--rw receiver-identity {receiver-identity}?
                 +--rw cert-maps
                    +--rw cert-to-name* [id]
                       +--rw id             uint32
                       +--rw fingerprint    x509c2n:tls-fingerprint
                       +--rw map-type       identityref
                       +--rw name           string
```

7.2. YANG module

The YANG module imports Common YANG Data Types [RFC9911], A YANG Data Model for SNMP Configuration [RFC7407], Subscription to YANG Notifications [RFC8639], YANG Data Structure Extensions [RFC8791], and YANG Groupings for HTTP Clients and HTTP Servers [I-D.ietf-netconf-http-client-server]. It also defines the "receiver-capabilities" structure used as the response body of the capabilities GET request.

The YANG module is shown below.

```
<CODE BEGINS> file "ietf-https-notif-transport@2026-06-30.yang"
module ietf-https-notif-transport {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-https-notif-transport";
  prefix "hnt";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 9911: Common YANG Data Types.";
  }
```

```
  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: YANG Data Model for SNMP Configuration.";
  }

  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }

  import ietf-subscribed-notif-receivers {
    prefix snr;
    reference
      "RFC XXXX: An HTTPS-based Transport for YANG Notifications.";
  }

  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions.";
  }

  import ietf-http-client {
    prefix httpc;
    reference
      "RFC YYYY: YANG Groupings for HTTP Clients and HTTP Servers.";
  }

  organization
    "IETF NETCONF Working Group";

  contact
    "WG Web:   <http://datatracker.ietf.org/wg/netconf>
     WG List:  <netconf@ietf.org>

     Authors: Mahesh Jethanandani (mjethanandani at gmail dot com)
              Kent Watsen (kent plus ietf at watsen dot net)";

  description
    "This YANG module is implemented by Publishers that implement
     the 'ietf-subscribed-notifications' module defined in RFC 8639.

     This module augments a 'case' statement called 'https' into
     the 'choice' statement called 'transport-type' defined
     by the 'ietf-https-notif-transport' module defined in RFC XXXX.
```

```
        Copyright (c) 2024 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.
        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Revised BSD
        License set forth in Section 4.c of the IETF Trust's Legal
        Provisions Relating to IETF Documents
        (http://trustee.ietf.org/license-info).

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

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
        'MAY', and 'OPTIONAL' in this document are to be interpreted as
        described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

    revision "2026-06-30" {
      description
        "Initial Version.";
      reference
        "RFC XXXX: An HTTPS-based Transport for YANG Notifications.";
    }

    feature receiver-identity {
      description
        "Indicates that the server supports filtering notifications
         based on the receiver's identity derived from its TLS
         certificate.";
    }

    identity https {
      base sn:transport;
      base sn:configurable-encoding;
      description
        "HTTPS transport for notifications.";
    }

    grouping https-receiver-grouping {
      description
        "A grouping that may be used by other modules wishing to
         configure HTTPS-based notifications without using RFC 8639.";
      container capabilities-endpoint {
        uses httpc:http-client-grouping;
        description
          "Defines 'capabilities' resource endpoint.";
      }
```

```
      container relay-notification-endpoint {
        uses httpc:http-client-grouping;
        description
          "Defines 'relay-notification' resource endpoint.";
      }

      container receiver-identity {
        if-feature receiver-identity;
        description
          "Maps the receiver's TLS certificate to a local identity
           enabling access control to be applied to filter out
           notifications that the receiver may not be authorized
           to view.";
        container cert-maps {
          uses x509c2n:cert-to-name;
          description
            "The cert-maps container is used by a TLS-based HTTP
             server to map the HTTPS client's presented X.509
             certificate to a 'local' username. Specifically, the
             'name' field within the module is used along with
             'specified' identity to perform the match. If no
             matching and valid cert-to-name list entry is found,
             the publisher MUST close the connection, and MUST
             NOT send any notifications over it.";
          reference
            "RFC 7407: A YANG Data Model for SNMP Configuration.";
        }
      }
    }

    augment "/sn:subscriptions/snr:receiver-instances/" +
            "snr:receiver-instance/snr:transport-type" {
      case https {
        container https-receiver {
          description
            "The HTTPS receiver to send notifications to.";
          uses https-receiver-grouping;
        }
      }
      description
        "Augment the transport-type choice to include the 'https'
         transport.";
    }

    sx:structure receiver-capabilities {
      description
        "A structure defining the response body for the GET request
         to the 'capabilities' resource.  Using standard YANG
```

```
          encoding rules, when encoded as 'application/yang-data+xml'
          the root element is 'receiver-capabilities' in the namespace
          of this module, and when encoded as
          'application/yang-data+json' the top-level key is
          'ietf-https-notif-transport:receiver-capabilities'.";
        leaf-list receiver-capability {
          type inet:uri;
          description
            "A capability supported by the receiver.  A partial list
             of capabilities is defined in the 'Capabilities for HTTPS
             Notification Receivers' registry (RFC XXXX).  Additional
             custom capabilities MAY be defined.";
        }
      }
    }
    <CODE ENDS>
```

8. Security Considerations

The YANG modules specified in this document define a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446]. The NETCONF Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The YANG modules in this document make use of groupings that are defined in YANG Groupings for HTTP Clients and HTTP Servers [I-D.ietf-netconf-http-client-server], YANG Groupings for TLS Clients and TLS Servers [RFC9645], and A YANG Data Model for SNMP Configuration [RFC7407]. Please see the Security Considerations section of those documents for considerations related to sensitivity and vulnerability of the data nodes defined in them. Additionally, the parameters defined in the tls-client-grouping in the ietf-tls-client module should follow the recommendations specified in Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). [RFC9325]

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

* The "path" node in "ietf-subscribed-notif-receivers" module can be modified by a malicious user to point to an invalid URI. Worse still, it could point the URI of their choosing, exploit the vulnerable client, and if redirects are followed to the same URI, track its usage.

  The container "receiver-identity" contains nodes like "cert-maps" that are used by the HTTP server to map to the HTTPS client's certificate to a 'local' username. An unintended modification of these nodes will result in new connection requests be denied.

Some of the readable data nodes in the YANG modules 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. The model does not define any readable subtrees and data nodes that are particularly sensitive or vulnerable.

Some of the RPC operations in the YANG modules may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. The model does not define any RPC operations.

9. IANA Considerations

9.1. The "IETF XML" Registry

This document registers two URIs in the "ns" subregistry of the "IETF XML" registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

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

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

9.2. The "YANG Module Names" Registry

This document registers two YANG modules in the "YANG Module Names" registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:

```
name:      ietf-subscribed-notif-receivers
namespace: urn:ietf:params:xml:ns:yang:ietf-subscribed-notif-receivers
prefix:    snr
reference: RFC XXXX

name:      ietf-https-notif-transport
namespace: urn:ietf:params:xml:ns:yang:ietf-https-notif-transport
prefix:    hnt
reference: RFC XXXX
```

9.3. Registration of 'yang-notif' URN Sub-namespace

This document requests that IANA register a new URN Sub-namespace within the "IETF URN Sub-namespace for Registered Protocol Parameter Identifiers" registry defined in [RFC3553].

```
Registry Name: yang-notif
Specification: RFC XXXX
Repository: "YANG Notifications" registry
```

9.4. Registration of 'https' URN Sub-namespace

This document requests that IANA register a new URN Sub-namespace within the "YANG Notifications" registry group defined in [RFC3553].

```
Registry Name: https-capability
Specification: RFC XXXX
Repository: "Capabilities for HTTPS Notification Receivers" registry
```

The following note shall be at the top of the registry:

```
This registry defines capabilities that can be
supported by HTTPS-based notification receivers.
```

The fields for each registry are:

* URN
  - The name of the URN (required).
  - The URN must conform to the syntax described by [RFC8141].

- The URN must begin with the string "urn:ietf:params:yang-notif:https-capability".

* Reference
  - The RFC that defined the URN.
  - The RFC must be in the form "RFC <Number>: <Title>.
* Description
  - An arbitrary description of the capability.
  - The description should be no more than a few sentences.
  - The description is to be in English, but may contain UTF-8 characters as may be needed in some cases.

The update policy is "RFC Required".

Following is the initial assignment for this registry:

```
Record:
   URN:         urn:ietf:params:yang-notif:https-capability:encoding:json
   Reference:   RFC XXXX:An HTTPS-based Transport for YANG Notifications
   Description: Identifies support for JSON-encoded notifications.

Record:
   URN:         urn:ietf:params:yang-notif:https-capability:encoding:xml
   Reference:   RFC XXXX:An HTTPS-based Transport for YANG Notifications
   Description: Identifies support for XML-encoded notifications.

Record:
   URN:         urn:ietf:params:yang-notif:https-capability:sub-notif
   Reference:   RFC XXXX:An HTTPS-based Transport for YANG Notifications
   Description: Identifies support for state machine described in
                RFC 8639, enabling the publisher to send, e.g., the
                "subscription-started" notification.
Record:
   URN:         urn:ietf:params:yang-notif:https-capability:rfc5277-notif
   Reference:   RFC XXXX:An HTTPS-based Transport for YANG Notifications
   Description: Identifies support for receiving legacy RFC 5277
                XML notifications, transmitted using "application/xml"
                as the Content-Type header value.
```

10. References

10.1. Normative references

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

[RFC3553] Mealling, M., Masinter, L., Hardie, T., and G. Klyne, "An IETF URN Sub-namespace for Registered Protocol Parameters", BCP 73, RFC 3553, DOI 10.17487/RFC3553, June 2003, <https://www.rfc-editor.org/info/rfc3553>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/info/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/info/rfc6020>.

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

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

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <https://www.rfc-editor.org/info/rfc7407>.

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

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

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

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

[RFC8791] Bierman, A., BjÃ¶rklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <https://www.rfc-editor.org/info/rfc8791>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/info/rfc9112>.

[RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

[RFC9645] Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", RFC 9645, DOI 10.17487/RFC9645, October 2024, <https://www.rfc-editor.org/info/rfc9645>.

[RFC9911] SchÃ¶nwÃ¤lder, J., Ed., "Common YANG Data Types", RFC 9911, DOI 10.17487/RFC9911, December 2025, <https://www.rfc-editor.org/info/rfc9911>.

[I-D.ietf-netconf-http-client-server] Watsen, K., "YANG Groupings for HTTP Clients and HTTP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-http-client-server-33, 4 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-http-client-server-33>.

10.2. Informative references

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

Appendix A. Configuration Examples

This non-normative section shows two examples for how the "ietf-https-notif-transport" module can be used to configure a publisher to send notifications to a receiver.

In both examples, the publisher, being an HTTPS client, is configured to send notifications to a receiver.

A.1. Using Subscribed Notifications (RFC 8639)

This example shows how an RFC 8639 [RFC8639] based publisher can be configured to send notifications to a receiver.

```
================ NOTE: '\' line wrapping per RFC 8792 ================

<?xml version="1.0" encoding="UTF-8"?>
<subscriptions
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications\
">
  <receiver-instances
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notif-recei\
vers">
    <receiver-instance>
      <name>global-receiver-def</name>
      <https-receiver
          xmlns="urn:ietf:params:xml:ns:yang:ietf-https-notif-transp\
```

```
   ort"
            xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-\
   to-name">
          <capabilities-endpoint>
            <uri>
              <scheme>https</scheme>
              <authority>
                <host>receiver.example.com</host>
                <port>443</port>
              </authority>
              <path>some-path</path>
            </uri>
            <tls-client-parameters>
              <server-authentication>
                <ca-certs>
                  <inline-definition>
                    <certificate>
                      <name>Server Cert Issuer #1</name>
                      <cert-data>base64encodedvalue==</cert-data>
                    </certificate>
                  </inline-definition>
                </ca-certs>
              </server-authentication>
            </tls-client-parameters>
          </capabilities-endpoint>
          <relay-notification-endpoint>
            <uri>
              <scheme>https</scheme>
              <authority>
                <host>receiver.example.com</host>
                <port>443</port>
              </authority>
              <path>some-path</path>
            </uri>
            <tls-client-parameters>
              <server-authentication>
                <ca-certs>
                  <inline-definition>
                    <certificate>
                      <name>Server Cert Issuer #1</name>
                      <cert-data>base64encodedvalue==</cert-data>
                    </certificate>
                  </inline-definition>
                </ca-certs>
              </server-authentication>
            </tls-client-parameters>
          </relay-notification-endpoint>
          <receiver-identity>
```

```
            <cert-maps>
              <cert-to-name>
                <id>1</id>
                <fingerprint>11:0A:05:11:00</fingerprint>
                <map-type>x509c2n:san-any</map-type>
              </cert-to-name>
            </cert-maps>
          </receiver-identity>
        </https-receiver>
      </receiver-instance>
    </receiver-instances>
    <subscription>
      <id>6666</id>
      <transport xmlns:ph="urn:ietf:params:xml:ns:yang:ietf-https-noti\
  f-transport">ph:https</transport>
      <stream-subtree-filter>
        <some-subtree-filter/>
      </stream-subtree-filter>
      <stream>some-stream</stream>
      <receivers>
        <receiver>
          <name>subscription-specific-receiver-def</name>
          <receiver-instance-ref xmlns="urn:ietf:params:xml:ns:yang:ie\
  tf-subscribed-notif-receivers">global-receiver-def</receiver-instanc\
  e-ref>
        </receiver>
      </receivers>
    </subscription>
  </subscriptions>
  <truststore xmlns="urn:ietf:params:xml:ns:yang:ietf-truststore">
    <certificate-bags>
      <certificate-bag>
        <name>explicitly-trusted-server-ca-certs</name>
        <description>
          Trust anchors (i.e. CA certs) that are used to
          authenticate connections to receivers.  Receivers
          are authenticated if their certificate has a chain
          of trust to one of these CA certificates.
          certificates.
        </description>
        <certificate>
          <name>ca.example.com</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
        <certificate>
          <name>Fred Flintstone</name>
          <cert-data>base64encodedvalue==</cert-data>
        </certificate>
```

```
      </certificate-bag>
    </certificate-bags>
  </truststore>
```

A.2. Not Using Subscribed Notifications

In the case that it is desired to use HTTPS-based notifications outside of Subscribed Notifications, an application-specific module would need to define the configuration for sending the notification.

Following is an example module. Note that the module "uses" the "https-receiver-grouping" grouping from the "ietf-https-notif-transport" module.

```
module example-custom-module {
  yang-version 1.1;
  namespace "http://example.com/example-custom-module";
  prefix "custom";

  import ietf-https-notif-transport {
    prefix "hnt";
    reference
      "RFC XXXX: An HTTPS-based Transport for YANG Notifications.";
  }

  organization
    "Example, Inc.";

  contact
    "Support at example.com";

  description
    "Example of module not using Subscribed Notifications module.";

  revision "2026-06-30" {
    description
      "Initial Version.";
    reference
      "RFC XXXX: An HTTPS-based Transport for YANG Notifications.";
  }

  container example-module {
    description
      "Example of using HTTPS notif without having to
       implement Subscribed Notifications.";

    container https-receivers {
      description
```

```
          "A container of all HTTPS notif receivers.";
        list https-receiver {
          key "name";
          description
            "A list of HTTPS notif receivers.";
          leaf name {
            type string;
            description
              "A unique name for the https-notif receiver.";
          }
          uses hnt:https-receiver-grouping;
        }
      }
    }
  }
```

Following is what the corresponding configuration looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<example-module xmlns="http://example.com/example-custom-module">
  <https-receivers>
    <https-receiver>
      <name>foo</name>
      <capabilities-endpoint>
        <uri>
          <scheme>https</scheme>
          <authority>
            <host>receiver.example.com</host>
            <port>443</port>
          </authority>
          <path>some-path</path>
        </uri>
        <tls-client-parameters>
          <server-authentication>
            <ca-certs>
              <inline-definition>
                <certificate>
                  <name>Server Cert Issuer #1</name>
                  <cert-data>base64encodedvalue==</cert-data>
                </certificate>
              </inline-definition>
            </ca-certs>
          </server-authentication>
        </tls-client-parameters>
      </capabilities-endpoint>
      <relay-notification-endpoint>
        <uri>
          <scheme>https</scheme>
```

```
            <authority>
              <host>receiver.example.com</host>
              <port>443</port>
            </authority>
            <path>some-path</path>
          </uri>
          <tls-client-parameters>
            <server-authentication>
              <ca-certs>
                <inline-definition>
                  <certificate>
                    <name>Server Cert Issuer #1</name>
                    <cert-data>base64encodedvalue==</cert-data>
                  </certificate>
                </inline-definition>
              </ca-certs>
            </server-authentication>
          </tls-client-parameters>
        </relay-notification-endpoint>
      </https-receiver>
    </https-receivers>
  </example-module>
```

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by first name): Eric Voit, Henning Rogge, Martin Bjorklund, Reshad Rahman, and Rob Wilton.

In addition, the authors would also like to thank Quifang Ma for providing thoughtful comments as part of shepherd writeup.

Authors' Addresses

Mahesh Jethanandani
Kloud Services
Email: mjethanandani@gmail.com

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net

Network Configuration
Internet-Draft
Intended status: Standards Track
Expires: 7 January 2027

B. M. Chittapragada
S. Bhat
V. T. Rao
H. Arshad
M. P. Tahiliani
National Institute of Technology Karnataka

6 July 2026

CBOR Encoding for HTTPS-based YANG Notifications Transport
draft-ietf-netconf-https-notif-cbor-01

Abstract

This document extends [I-D.draft-ietf-netconf-https-notif] by introducing CBOR encoding for YANG notifications over HTTPS transport in addition to the existing JSON and XML encoding schemes.

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://MeherRushi.github.io/draft-ietf-netconf-https-notif-cbor/draft-ietf-netconf-https-notif-cbor.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-netconf-https-notif-cbor/.

Discussion of this document takes place on the Network Configuration 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/MeherRushi/draft-ietf-netconf-https-notif-cbor.

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/.

Table of Contents

1. Introduction

CBOR offers an efficient and compact representation of YANG.

This document introduces a CBOR [RFC8949] encoding scheme for event notifications over HTTPS by using the framework proposed in [I-D.draft-ietf-netconf-https-notif] which supports transfer of YANG notifications over HTTPS using JSON and XML encoding schemes.

In [I-D.draft-ietf-netconf-https-notif], the capabilities HTTP-target resource allows a publisher to retrieve supported encoding formats via GET requests, while the relay-notification resource enables the publisher to send YANG notifications via POST requests. These requests and responses use different content types based on the selected encoding scheme. This document defines support for CBOR encoding, in addition to the JSON and XML encodings defined in [I-D.draft-ietf-netconf-https-notif].

Examples of the GET and POST request and reply encoded in CBOR are also provided.

2. Terminology

This document uses the following terms defined in Sections 2, 3, and 4 of [I-D.draft-ietf-netconf-https-notif]:

* Capabilities Resource

* Relay-Notification

* Event Notification

The following term(s) are defined in Subscription to YANG Notifications [RFC8639]:

* Publisher

* Receiver

* Subscribed Notifications

The following term(s) are defined in Encoding of Data Modeled with YANG in the Concise Binary Object Representation (CBOR) [RFC9254]:

* Diagnostic Notation

* YANG Schema Item iDentifier (or "YANG SID" or simply "SID"): 63-bit unsigned integer used to identify different YANG items.

3. CBOR Encoding of the notification(s)

YANG notifications can be encoded in CBOR using Names or SIDs in keys. Notifications encoded using names is similar to JSON encoding as defined in Section 4.1 of [I-D.draft-ietf-netconf-https-notif]. Notification encoded using YANG-SIDs replaces the names of the keys of the CBOR encoded message with a 63 bit unsigned integer. In this case, the term 'SID' is defined in Section 3.2 of [RFC9254], and the keys of the encoded data use SID value as mentioned in 4.3.2 of this document.

The "application/yang-data+cbor" media type used throughout this document is defined in Section 9 of [RFC9254]. That registration also defines the "id" parameter: "application/yang-data+cbor; id=name" for the name-based keys described in this document, and "application/yang-data+cbor; id=sid" for the SID-based keys.

3.1. Capabilities Request

The publisher sends a request to the receiver to learn its capabilities. In the below example, the â\200\234Acceptâ\200\235 states that the publisher wants to receive the capabilities response in CBOR but if not supported then in XML or JSON in that order.

```
GET /some/path/capabilities HTTP/1.1
   Host: example.com
   Accept: application/yang-data+cbor; id=name, application/yang-data+xml;q=0.
9, application/yang-data+json;q=0.5
```

3.2. Capabilities Response

If the receiver is able to reply using â\200\234application/yang-data+cborâ\200\235 and assuming it is only capable of receiving CBOR encoded messages the response would look like this

3.2.1. CBOR using names as keys (receiver accepts CBOR only)

```
   HTTP/1.1 200 OK
   Date: Tue, 4 March 2025 20:33:30 GMT
   Server: example-server
   Cache-Control: no-cache
   Content-Type: application/yang-data+cbor; id=name
```

Diagnostic Notation:

```
      {
      "ietf-https-notif-transport:receiver-capabilities": {
        "receiver-capability": [
          "urn:ietf:params:yang-notif:https-capability:encoding:cbor"
          ]
        }
      }
```

CBOR Encoding:

```
A1                                      # map(1)
   78 30                                # text(48)
      696574662D68747470732D6E6F7469662D7472616E73706F72743A7265365697665722D636
1706162696C6974696573 # "ietf-https-notif-transport:receiver-capabilities"
   A1                                   # map(1)
      73                                # text(19)
         7265365697665722D6361706162696C697479 # "receiver-capability"
      81                                # array(1)
         78 39                          # text(57)
            75726E3A696574663A706172616D733A79616E672D6E6F7469663A68747470732D636
1706162696C6974793A656E636F64696E673A63626F72 # "urn:ietf:params:yang-notif:https
-capability:encoding:cbor"
```

If the receiver is able to reply using â\200\234application/yang-data+cborâ
\200\235
and assuming it is not capable of receiving cbor, but can receive
both json and xml notifications:

3.2.2. CBOR using names as keys (receiver accepts JSON and XML)

```
      HTTP/1.1 200 OK
      Date: Tue, 4 March 2025 20:33:30 GMT
      Server: example-server
      Cache-Control: no-cache
      Content-Type: application/yang-data+cbor; id=name
```

Diagnostic Notation:

```
      {
      "ietf-https-notif-transport:receiver-capabilities": {
        "receiver-capability": [
          "urn:ietf:params:yang-notif:https-capability:encoding:json",
          "urn:ietf:params:yang-notif:https-capability:encoding:xml"
          ]
        }
      }
```

CBOR Encoding:

```
A1                                      # map(1)
   78 30                                # text(48)
      696574662D68747470732D6E6F7469662D7472616E73706F72743A7265365697665722D636
1706162696C6974696573 # "ietf-https-notif-transport:receiver-capabilities"
   A1                                   # map(1)
      73                                # text(19)
         7265365697665722D6361706162696C697479 # "receiver-capability"
      82                                # array(2)
         78 39                          # text(57)
            75726E3A696574663A706172616D733A79616E672D6E6F7469663A68747470732D636
1706162696C6974793A656E636F64696E673A6A736F6E # "urn:ietf:params:yang-notif:https
-capability:encoding:json"
         78 38                          # text(56)
            75726E3A696574663A706172616D733A79616E672D6E6F7469663A68747470732D636
1706162696C6974793A656E636F64696E673A786D6C # "urn:ietf:params:yang-notif:https-c
apability:encoding:xml"
```

If the receiver cannot encode the capabilities response in "application/yang-data+cbor" but is itself only able to receive CBOR-encoded notifications, it replies using JSON (or XML) while still advertising the CBOR capability:

```
HTTP/1.1 200 OK
Date: Tue, 4 March 2025 20:33:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+json

{
"ietf-https-notif-transport:receiver-capabilities": {
  "receiver-capability": [
    "urn:ietf:params:yang-notif:https-capability:encoding:cbor"
    ]
  }
}
```

3.3. Relay Notification request

The publisher sends an HTTP POST request to the "relay-notification" resource on the receiver with the "Content-Type" header set to "application/yang-data+cbor" in case the receiver is CBOR capable and a body containing the notification encoded in CBOR. The "id" parameter of the media type indicates whether the keys are encoded as names ("id=name") or as SIDs ("id=sid").

3.3.1. CBOR encoding using names as keys

```
POST /some/path/relay-notification HTTP/1.1
   Host: example.com
   Content-Type: application/yang-data+cbor; id=name
```

Diagnostic Notation:

```
      {
        "ietf-https-notif:notification": {
          "eventTime": "2013-12-21T00:01:00Z",
          "example-mod:event" : {
            "event-class" : "fault",
            "reporting-entity" : { "card" : "Ethernet0" },
            "severity" : "major"
          }
        }
      }
```

CBOR Encoding:

```
A1                                      # map(1)
   78 1D                                # text(29)
      696574662D68747470732D6E6F7469663A6E6F74696669636174696F6E # "ietf-https-no
tif:notification"
   A2                                   # map(2)
      69                                # text(9)
         6576656E7454696D65             # "eventTime"
      74                                # text(20)
         323031332D31322D32315430303A30313A30305A # "2013-12-21T00:01:00Z"
      71                                # text(17)
         6578616D706C652D6D6F643A6576656E74 # "example-mod:event"
      A3                                # map(3)
         6B                             # text(11)
            6576656E742D636C617373      # "event-class"
         65                             # text(5)
            6661756C74                  # "fault"
         70                             # text(16)
            7265706F7274696E672D656E74697479 # "reporting-entity"
         A1                             # map(1)
            64                          # text(4)
               63617264                 # "card"
            69                          # text(9)
               45746865726E657430       # "Ethernet0"
         68                             # text(8)
            7365766572697479            # "severity"
         65                             # text(5)
            6D616A6F72                  # "major"
```

3.3.2. CBOR encoding using SIDs as keys

```
   POST /some/path/relay-notification HTTP/1.1
      Host: example.com
      Content-Type: application/yang-data+cbor; id=sid
```

Diagnostic Notation:

```
{
  2601: {                          / ietf-https-notif:notification (SID 2601) /
    1: "2013-12-21T00:01:00Z",     / eventTime (SID 2602) /
    57400: {                       / example-mod:event (SID 60001) /
      1: "fault",                  / event-class (SID 60002) /
      2: {                         / reporting-entity (SID 60003) /
        1: "Ethernet0"             / card (SID 60004) /
      },
      4: "major"                   / severity (SID 60005) /
    }
  }
}
```

The above assumes that the YANG modules for the notification envelope and for the event have corresponding .sid files with the following entries:

```
   "item": [
         {
           "namespace": "module",
           "identifier": "ietf-https-notif",
           "sid": "2600"
         },
         {
           "namespace": "data",
           "identifier": "/ietf-https-notif:notification",
           "sid": "2601"
         },
         {
           "namespace": "data",
           "identifier": "/ietf-https-notif:notification/eventTime",
           "sid": "2602"
         },
         {
           "namespace": "module",
           "identifier": "example-mod",
           "sid": "60000"
         },
         {
           "namespace": "data",
           "identifier": "/example-mod:event",
           "sid": "60001"
         },
         {
           "namespace": "data",
           "identifier": "/example-mod:event/event-class",
           "sid": "60002"
         },
         {
           "namespace": "data",
           "identifier": "/example-mod:event/reporting-entity",
           "sid": "60003"
         },
         {
           "namespace": "data",
           "identifier": "/example-mod:event/reporting-entity/card",
           "sid": "60004"
         },
         {
           "namespace": "data",
           "identifier": "/example-mod:event/severity",
           "sid": "60005"
         }
       ]
```

The SID values shown above are illustrative examples only; they are not SIDs registered for any real YANG module. A deployment uses the SIDs actually assigned to the modules involved.

Keys are encoded as SID deltas relative to the reference SID of the enclosing map, as defined in Section 3.2 of [RFC9254]. Because "example-mod:event" is defined in a different module than its parent, its delta is large (60001 - 2601 = 57400); alternatively, it MAY be encoded as an absolute SID using CBOR tag 47.

CBOR Encoding:

```
A1                                      # map(1)
   19 0A29                              # unsigned(2601)
   A2                                   # map(2)
      01                                # unsigned(1)
      74                                # text(20)
         323031332D31322D32315430303A30313A30305A # "2013-12-21T00:01:00Z"
      19 E038                           # unsigned(57400)
      A3                                # map(3)
         01                             # unsigned(1)
         65                             # text(5)
            6661756C74                  # "fault"
         02                             # unsigned(2)
         A1                             # map(1)
            01                          # unsigned(1)
            69                          # text(9)
               45746865726E657430       # "Ethernet0"
         04                             # unsigned(4)
         65                             # text(5)
            6D616A6F72                  # "major"
```

3.4. Relay Notification Response

The response on success SHOULD be from the 2XX class of codes. In case of corrupted or malformed event, the response SHOULD be an appropriate HTTP error response.

3.5. Legacy RFC 5277 Notifications

[I-D.draft-ietf-netconf-https-notif] defines optional support for sending legacy notifications, as defined in NETCONF Event Notifications [RFC5277], using the "application/xml" media type. This legacy mode exists for interoperability with publishers that are not aware of YANG media types; a receiver advertises support for it using the "urn:ietf:params:yang-notif:https-capability:rfc5277-notif" capability.

The CBOR encoding defined in this document applies to YANG-modeled notifications, including the eventTime envelope, and is advertised using the "urn:ietf:params:yang-notif:https-capability:encoding:cbor" capability registered by this document. Because [I-D.draft-ietf-netconf-https-notif] defines the legacy RFC 5277 mode as XML-only, this document does not define a CBOR variant of it. A publisher that is able to produce CBOR is YANG-aware and therefore uses the CBOR encoding described here rather than the legacy mode.

3.6. Implementation Status

This section records the status of known implementations of the specification defined by this document at the time of posting. The information is provided to assist the IETF in evaluating the maturity and implementability of the specification. This section will be removed prior to publication as an RFC.

3.7. Implementation: HTTPS Notification CBOR Draft Implementation

* _Organization_: National Institute of Technology Karnataka

* _Implementation Name / Web Page_: HTTPS Notification CBOR Draft Implementation https://github.com/MeherRushi/https-notif-draft-impl (https://github.com/MeherRushi/https-notif-draft-impl)

* _Description_: This implementation provides a Python-based prototype of the mechanism defined in this document for transporting YANG notifications over HTTPS using JSON, XML and CBOR encoding. It supports name-based CBOR encoding and includes basic publisher and receiver roles to demonstrate end-to-end message exchange.

* _Maturity Level_: Prototype

* _Coverage_:

  - Capabilities discovery via HTTP GET to /capabilities

  - Event publication via HTTP POST to /relay-notification

  - Support for name-based CBOR encoding as described in this document

* _Version Compatibility_: The implementation is based on draft-ietf-netconf-https-notif-16 and draft-ietf-netconf-https-notif-cbor-00.

* _Licensing_: Freely distributable under an MIT-style license.

* _Implementation Experience_:
  - Developed and demonstrated at IETF 121 and 122 Hackathon.
  - Worked toward enabling CBOR encoding in the libyang library as part of the hackathon effort (slides (https://datatracker.ietf.org/meeting/123/materials/slides-123-hackathon-sessd-adding-cbor-support-in-libyang-00)).
  - Evaluated CBOR efficiency compared to JSON and XML in constrained environments.
  - Built tooling to simulate and measure notification transfer behavior over varying network conditions.
  - Diagnostic encoding examples used for validation of CBOR structures.
* _Contact Information_:
  - Bharadwaja Meherrushi Chittapragada (meher.211cs216@nitk.edu.in)
  - Vartika T Rao (vartikatrao.211it077@nitk.edu.in)
  - Siddharth Bhat (sidbhat.211ee151@nitk.edu.in)
  - Hayyan Arshad (hayyanarshad.211cs222@nitk.edu.in)
* _Last Updated_: July 24, 2025

4. Security Considerations

Addition of the CBOR encoding introduces no specific security exposures or risks other than the ones mentioned in [RFC9254] and [I-D.draft-ietf-netconf-https-notif] (An HTTPS-based Transport for YANG Notifications).

5. IANA Considerations

This document requests that IANA include an additional entry in the â\200\234Capabilities for HTTPS Notification Receiversâ\200\235 registry, defined in [I-D.draft-ietf-netconf-https-notif]. The following entry is added:

Record:
URN: urn:ietf:params:yang-notif:https-capability:encoding:cbor
Reference: RFC XXXX: CBOR Encoding for HTTPS-based YANG Notifications Transport
Description: Identifies support for CBOR-encoded notifications.

6. Normative References

[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-16, 30 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-16>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/rfc/rfc5277>.

[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>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[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>.

Acknowledgments

The authors acknowledge the support of Kent Watsen and Mahesh Jethanandani, the authors of [I-D.draft-ietf-netconf-https-notif] for their guidance and support provided to draft this document.

Authors' Addresses

Bharadwaja Meherrushi Chittapragada
National Institute of Technology Karnataka
Email: meherrushi2@gmail.com

Siddharth Bhat
National Institute of Technology Karnataka
Email: siddharth.bhat10@gmail.com

Vartika T Rao
National Institute of Technology Karnataka
Email: vartikatrao@gmail.com

Hayyan Arshad
National Institute of Technology Karnataka
Email: hayyanhamnah@gmail.com

Mohit P. Tahiliani
National Institute of Technology Karnataka
Email: tahiliani@nitk.edu.in

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 7 January 2027

K. Watsen
Watsen Networks
Q. Wu
Huawei Technologies
P. Andersson
Ionio Systems
O. Hagsand
SUNET
H. Li
Hewlett Packard Enterprise

6 July 2026

List Pagination for YANG-driven Protocols
draft-ietf-netconf-list-pagination-13

Abstract

In some circumstances, instances of YANG modeled "list" and "leaf-list" nodes may contain numerous entries. Retrieval of all the entries can lead to inefficiencies in the server, the client, and the network in between.

This document defines a model for list pagination that can be implemented by YANG-driven management protocols such as NETCONF and RESTCONF. The model supports paging over optionally filtered and/or sorted entries. The solution additionally enables servers to constrain query expressions on some "config false" lists or leaf-lists.

Table of Contents

1. Introduction

YANG modeled "list" and "leaf-list" nodes may contain a large number of entries. For instance, there may be thousands of entries in the configuration for network interfaces or access control lists. And time-driven logging mechanisms, such as an audit log or a traffic log, can contain millions of entries.

Retrieval of all the entries can lead to inefficiencies (e.g., long loading time, memory overconsuming, or crash) in the server, the client, and the network in between. For instance, consider the following:

* A client may need to filter and/or sort list entries in order to, e.g., present the view requested by a user.

* A server may need to iterate over many more list entries than needed by a client.

* A network may need to convey more data than needed by a client.

Optimal global resource utilization is obtained when clients are able to cherry-pick just that which is needed to support the application-level business logic.

This document defines a generic model for list pagination that can be implemented by YANG-driven management protocols such as NETCONF [RFC6241] and RESTCONF [RFC8040]. How the NETCONF and RESTCONF protocols support list pagination or protocols mapping is described in [I-D.ietf-netconf-list-pagination-nc] and [I-D.ietf-netconf-list-pagination-rc], respectively.

The model presented in this document supports paging over optionally filtered and/or sorted entries. Server-side filtering and sorting is ideal as servers can leverage indexes maintained by a backend storage layer to accelerate queries.

1.1. Terminology

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

The following terms are defined in [RFC7950] and are not redefined here: client, data model, data tree, feature, extension, module, leaf, leaf-list, and server.

1.2. Conventions

Various examples in this document use "BASE64VALUE=" as a placeholder value for binary data that has been base64 encoded (per Section 9.8 of [RFC7950]). This placeholder value is used because real base64 encoded structures are often many lines long and hence distracting to the example being presented.

1.3. Adherence to the NMDA

This document is compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. The "ietf-list-pagination" module defines the "sort" feature, annotations and identities for protocol signaling, a grouping for the list paginaton query parameters, and also augments leafs into a "config false" node defined by the "ietf-system-capabilities" module.

2. Solution Overview

The solution presented in this document broadly entails a client sending a query to a server targeting a specific list or leaf-list including optional parameters guiding which entries should be returned.

Furthermore the solution is intended to leverage underlying database system capabilities, e.g. fast lookups relying on indexes, using efficient built-in selection and pagination functions. However, since there are many different database systems and configurations, the solution defines a common subset of functionality broadly available. It is also possible that a datastore’s underlying database system is federated, i.e. consists of several different

database systems to provide one datastore. In order to form a general solution, the possibility to configure and tune what components are available is presented.

A secondary aspect of this solution entails a client sending a query parameter to a server guiding how descendant lists and leaf-lists should be returned. This parameter may be used on any target node, not just "list" and "leaf-list" nodes.

Clients detect a server's support for list pagination via an entry for the "ietf-list-pagination" module (defined in Section 4) in the server's YANG Library [RFC8525] response.

Relying on client-provided query parameters ensures servers remain backward compatible with legacy clients.

3. Solution Details

This section is composed of the following subsections:

* Section 3.1 defines five query parameters clients may use to page through the entries of a single list or leaf-list in a data tree.

* Section 3.2 defines one query parameter that clients may use to affect the content returned for descendant lists and leaf-lists.

* Section 3.3 defines per schema-node tags enabling servers to indicate which "config false" lists are constrained and how they may be interacted with.

3.1. Query Parameters for a Targeted List or Leaf-List

The five query parameters presented in this section are listed in processing order. This processing order is logical, efficient, and matches the processing order implemented by database systems, such as SQL.

The order is as follows: a server first processes the "where" parameter (see Section 3.1.1), then the "sort-by" parameter (see Section 3.1.2), then a combination of the "direction" parameter (see Section 3.1.4), with either the "offset" parameter (see Section 3.1.5) or the "cursor" parameter (see Section 3.1.6), and lastly the "limit" parameter (see Section 3.1.7).

The sorting can furthermore be configured with a locale for sorting. This is done by setting the "locale" parameter (see Section 3.1.3).

The feature "sort" is used to indicate support for the query parameters "locale" and "sort-by".

3.1.1. The "where" Query Parameter

Description
: The "where" query parameter specifies a filter expression that result-set entries must match. If the string "unfiltered" is supplied, no entries are filtered from the working result-set.

Default Value
: If this query parameter is unspecified, the default value is "unfiltered".

Allowed Values
: The allowed values are the XPath 1.0 expressions defined in Section 10 of [RFC7950]. The XPath context follows Section 6.4.1 of [RFC7950] and, in addition, the context node is the target of the query, the accessible tree can be specific entry of "config true" lists and leaf-lists or "config false" lists and leaf-lists. Details (such as prefix bindings) are further defined at the protocol level, e.g., [I-D.ietf-netconf-list-pagination-nc] and [I-D.ietf-netconf-list-pagination-rc].

  If an XPath expression references a node identifier that does not exist in the schema, or use undefined prefixes, or is optional or conditional in the schema, the filter evaluates to false and nothing is filtered from the result-set.

  If the value "unfiltered" is supplied, no filtering is done.

Error Handling
: If the specified XPath expression is invalid, then an error MUST be returned with the error-type value "application" and error-tag value "invalid-value".

  For constrained "config false" lists or leaf-lists, it is an error if the specified node identifier point to a node without the "indexed" leaf set to "true" (see Section 3.3.2).

Conformance
: When the "where" feature is enabled, the "where" query parameter MUST be supported for all "config true" lists and leaf-lists and SHOULD be supported for "config false" lists and leaf-lists. The supplied expression is allowed to be of arbitrary complexity. Servers MAY disable the support for some or all "config false" lists and leaf-lists as described in Section 3.3.2.

3.1.2. The "sort-by" Query Parameter

Description
: The "sort-by" query parameter indicates the node in the working result-set (i.e., after the "where" parameter has been applied) that entries should be sorted by. Sorts are in ascending order (e.g., '1' before '9', 'a' before 'z', etc.). Missing values are sorted to the end (e.g., after all nodes having values). Sub-sorts are not supported.

Default Value
: If this query parameter is unspecified, then the list or leaf-list's default order is used, per the YANG "ordered-by" statement (see Section 7.7.7 of [RFC7950]).

Allowed Values
: The allowed values are node identifiers. Clients are allowed to request sorting by any data node within the result-set. If the specified node identifier is optional or conditional in the schema, the default strategy should be applied to entries without the sort-by value, i.e., place entries without the sort-by value after all entries that have a value.

Error Handling
: If the specified node identifier is invalid, then an error MUST be returned with the error-type value "application" and error-tag value "invalid-value".

  For constrained "config false" lists or leaf-lists, it is an error if the specified node identifier point to a node without the "indexed" leaf set to "true" (see Section 3.3.2).

Conformance
: When the "sort-by" feature is enabled, the "sort-by" query parameter MUST be supported for all "config true" lists and leaf-lists and SHOULD be supported for all "config false" lists and leaf-lists. Servers MAY disable the support for some or all "config false" lists and leaf-lists as described in Section 3.3.2.

3.1.3. The "locale" Query Parameter

Description
: The "locale" query parameter indicates what locale is used when sorting the result-set. Note that the "locale" query parameter is invalid to supply without also supplying the "sort-by" query parameter. If a query supplies "locale" and not "sort-by", error-type application and error-tag "invalid-value" is returned.

Default Value
: If this query parameter is unspecified, it is up to the server to select a locale for sorts. How the server chooses the locale used is out of scope for this document. The result-set includes the locale used by the server for sorts with a metadata value [RFC7952] called "locale".

Allowed Values
: The format is a free form string but SHOULD follow the language sub-tag format defined in [RFC5646]. An example is 'sv_SE'. Note that all locales are assumed to be UTF-8, since character encoding for YANG strings and all known YANG modelled encodings and protocols are required to be UTF-8 [RFC6241] [RFC7950] [RFC7951] [RFC8040]. A server MUST accept a known encoding with or without trailing ".UTF-8" and MAY emit an encoding with or without trailing ".UTF-8". This means a server must handle both e.g. "sv_SE" and "sv_SE.UTF-8" equally as input, and chooses how to emit used locale as output.

Error Handling
: If the specified locale is invalid, i.e. the locale is unknown to the server, then an error MUST be returned with the error-type value "application", error-tag value "invalid-value", and error-app-tag value "locale-unavailable".

Conformance
: When the "sort-by" parameter is specified, the "locale" query parameter MUST be supported for all "config true" lists and leaf-lists SHOULD be supported for "config false" lists and leaf-lists. Servers MAY disable the support for some or all "config false" lists and leaf-lists as described in Section 3.3.2.

3.1.4. The "direction" Query Parameter

Description
: The "direction" query parameter indicates how the entries in the working result-set (i.e., after the "sort-by" parameter has been applied) should be traversed.

Default Value
: If this query parameter is unspecified, the default value is "forwards".

Allowed Values
: The allowed values are:

forwards
: Return entries in the forwards direction. Also known as the "default" or "ascending" direction.

backwards
: Return entries in the backwards direction. Also known as the "reverse" or "descending" direction

Error Handling
: If the direction value is invalid, then an error MUST be returned with the error-type value "application" and error-tag value "invalid-value".

Conformance
: The "direction" query parameter MUST be supported for all lists and leaf-lists.

3.1.5. The "offset" Query Parameter

Description
: The "offset" query parameter indicates the number of entries in the working result-set (i.e., after the "direction" parameter has been applied) that should be skipped over when preparing the response. The "offset" query parameter MUST not be used together with "cursor" query parameter

Default Value
: If this query parameter is unspecified, then no entries in the result-set are skipped, the same as when the offset value '0' is specified.

Allowed Values
: The allowed values are unsigned integers.

Error Handling
: If the offset value is invalid, an error MUST be returned with the error-type value "application" and error-tag value "invalid-value".

  If the offset value exceeds the number of entries in the working result set, then the error also includes the error-app-tag value "offset-out-of-range".

Conformance
: The "offset" query parameter MUST be supported for all lists and leaf-lists.

3.1.6. The "cursor" Query Parameter

Description

The "cursor" query parameter indicates where to start the working result-set (i.e., after the "direction" parameter has been applied), the elements before the cursor are skipped over when preparing the response. Furthermore, a result set constrained with the "limit" query parameter includes metadata values [RFC7952] called "next" and "previous", which contains cursor values to the next and previous result-sets.

The next and previous cursor values are opaque index values for the underlying system's database, e.g. a key or other information needed to efficiently access the selected result-set. These "next" and "previous" metadata values also embed the query parameters from the initial query. E.g. if the initial query used the "where" query parameter, this is not supplied in subsequent requests. This means that when the cursor query parameter doesn't address the first element in a list, the only other allowed query parameters are "limit" and "sublist-limit". Using "limit" together with "cursor" affects both "next" and "previous" metadata values, by addressing the node the supplied amount of posts forward and backwards in the result-set. The cursor values are never cached.

Default Value

If this query parameter is unspecified, then no entries in the result-set are skipped, the same as when the cursor value "first" is specified.

Allowed Values

The allowed values are opaque encoded positions interpreted by the server to index an element in the list, e.g. a list key or other information to efficiently access the selected result-set.

If the value "first" is supplied, no skipping is done.

Error Handling

If the "cursor" RPC input parameter is not supported on the target node, then the error MUST be returned with error-type value "application" and error-tag value "operation-not-supported".

If the cursor value is unknown, i.e. the key does not exist, an error MUST be returned with the error-type value "application", error-tag value "invalid-value", and error-app-tag value "cursor-not-found".

Conformance

The "cursor" query parameter MUST be supported for all "config true" lists and SHOULD be supported for all "config false" lists. As the "cursor" query parameter support for "config false" lists is optional, the support MUST be signaled by the server per list.

Servers indicate that they support the "cursor" query parameter for a "config false" list node by having the "cursor-supported" leaf set to "true" in the "per-node-capabilities" node in the "ietf-system-capabilities" model.

Since leaf-lists might not have any unique values that can be indexed, the "cursor" query parameter is not relevant for leaf-lists. Consider the following leaf-list [1,1,2,3,5], which contains elements without uniquely indexable values. It would be possible to use the position, but then the solution would be equal to using the "offset" query parameter.

3.1.7. The "limit" Query Parameter

Description
: The "limit" query parameter limits the number of entries returned from the working result-set (i.e., after the "offset" parameter has been applied). Any list or leaf-list that is limited includes, somewhere in its encoding, a metadata value [RFC7952] called "remaining", a positive integer indicating the number of elements that were not included in the result-set by the "limit" operation, or the value "unknown" in case, e.g., the server determines that counting would be prohibitively expensive. If the string "unbounded" is supplied, there is no limit imposed on the result-set.

Default Value
: If this query parameter is unspecified, the number of entries that may be returned is unbounded.

Allowed Values
: The allowed values are unsigned 32 bit integers greater than or equal to 1, or the string "unbounded".

Error Handling
: If the limit value is invalid, i.e. not an unsigned 32-bit integer greater than or equal to 1 or the string "unbounded", then an error MUST be returned with the error-type value "application" and error-tag value "invalid-value".

Conformance
: The "limit" query parameter MUST be supported for all lists and leaf-lists.

3.2. Query Parameter for Descendant Lists and Leaf-Lists

Whilst this document primarily regards pagination for a list or leaf-list, it begs the question for how descendant lists and leaf-lists should be handled, which is addressed by the "sublist-limit" query parameter described in this section.

3.2.1. The "sublist-limit" Query Parameter

Description
: The "sublist-limit" parameter limits the number of entries returned for descendant lists and leaf-lists.

  Any descendant list or leaf-list limited by the "sublist-limit" parameter includes, somewhere in its encoding, a metadata value [RFC7952] called "remaining", a positive integer indicating the number of elements that were not included by the "sublist-limit" parameter, or the value "unknown" in case, e.g., the server determines that counting would be prohibitively expensive.

  When used on a list node, it only affects the list's descendant nodes, not the list itself, which is only affected by the parameters presented in Section 3.1.

Default Value
: If this query parameter is unspecified, the number of entries that may be returned for descendant lists and leaf-lists is unbounded.

Allowed Values
: The allowed values are positive integers.

Error Handling
: If the sublist-limit value is invalid, then an error MUST be returned with the error-type value "application" and error-tag value "invalid-value".

  .

Conformance
: The "sublist-limit" query parameter MUST be supported for all conventional nodes, including a datastore's top-level node (i.e., '/').

3.3. Constraints on "where" and "sort-by" for "config false" Lists

Some "config false" lists and leaf-lists may contain an enormous number of entries. For instance, a time-driven logging mechanism, such as an audit log or a traffic log, can contain millions of entries.

In such cases, "where" and "sort-by" expressions will not perform well if the server must bring each entry into memory in order to process it.

The server's best option is to leverage query-optimizing features (e.g., indexes) built into the backend database holding the dataset.

However, translating arbitrary "where" expressions and "sort-by" node identifiers into syntax supported by the backend database and/or query-optimizers may prove challenging, if not impossible, to implement.

Thusly this section introduces mechanisms whereby a server can:

1. Identify which "config false" lists and leaf-lists are constrained.

2. Identify what node-identifiers and expressions are allowed for the constrained lists and leaf-lists.

> Note: The pagination performance for "config true" lists and leaf-lists is not considered as servers must already be able to process them as configuration. Whilst some "config true' lists and leaf-lists may contain thousands of entries, they are well within the capability of server-side processing.

3.3.1. Identifying Constrained "config false" Lists and Leaf-Lists

Identification of which lists and leaf-lists are constrained occurs in the schema tree, not the data tree. However, as server abilities vary, it is not possible to define constraints in YANG modules defining generic data models.

In order to enable servers to identify which lists and leaf-lists are constrained, the solution presented in this document augments the data model defined by the "ietf-system-capabilities" module presented in [RFC9196].

Specifically, the "ietf-list-pagination" module (see Section 4) augments a boolean leaf node called "constrained" into the "per-node-capabilities" node defined in the "ietf-system-capabilities" module.

The "constrained" leaf MAY be specified for any "config false" list or leaf-list.

When a list or leaf-list is constrained:

* All parts of the XPath expressions are disabled unless explicitly enabled by Section 3.3.2.

* Node-identifiers used in "where" expressions and "sort-by" filters MUST have the "indexed" leaf set to "true" (see Section 3.3.2).

* For lists only, node-identifiers used in "where" expressions and "sort-by" filters MUST NOT descend past any descendant lists. This ensures that only indexes relative to the targeted list are used. Further constraints on node identifiers MAY be applied in Section 3.3.2.

3.3.2. Indicating the Constraints for "where" Filters and "sort-by" Expressions

This section identifies how constraints for "where" filters and "sort-by" expressions are specified. These constraints are valid only if the "constrained" leaf described in the previous section Section 3.3.1 has been set on the immediate ancestor "list" node or, for "leaf-list" nodes, on itself.

3.3.2.1. Indicating Filterable/Sortable Nodes

For "where" filters, an unconstrained XPath expressions may use any node in comparisons. However, efficient mappings to backend databases may support only a subset of the nodes.

Similarly, for "sort-by" expressions, efficient sorts may only support a subset of the nodes.

In order to enable servers to identify which nodes may be used in comparisons (for both "where" and "sort-by" expressions), the "ietf-list-pagination" module (see Section 4) augments a boolean leaf node called "indexed" into the "per-node-capabilities" node defined in the "ietf-system-capabilities" module (see [RFC9196]).

When a "list" or "leaf-list" node has the "constrained" leaf, only nodes having the "indexed" node set to "true" may be used in "where" and/or "sort-by" expressions. If no nodes have the "indexed" leaf set to "true", when the "constrained" leaf is set to "true", then "where" and "sort-by" expressions are disabled for that list or leaf-list.

4. The "ietf-list-pagination" Module

The "ietf-list-pagination" module is used by servers to indicate that they support pagination on YANG "list" and "leaf-list" nodes, and to provide an ability to indicate which "config false" list and/or "leaf-list" nodes are constrained and, if so, which nodes may be used in "where" and "sort-by" expressions.

4.1. Data Model Overview

The following tree diagram [RFC8340] illustrates the "ietf-list-pagination" module:

```
module: ietf-list-pagination

  augment /sysc:system-capabilities/sysc:datastore-capabilities
            /sysc:per-node-capabilities:
    +--ro constrained?        boolean
    +--ro indexed?            boolean
    +--ro cursor-supported?   boolean
```

Comments:

* As shown, this module augments three optional leafs into the "per-node-capabilities" node of the "ietf-system-capabilities" module.

* Not shown is that the module also defines an "md:annotation" statement named "remaining". This annotation may be present in a server's response to a client request containing either the "limit" (Section 3.1.7) or "sublist-limit" parameters (Appendix A.3.8).

4.2. Example Usage

4.2.1. Constraining a "config false" list

The following example illustrates the "ietf-list-pagination" module's augmentations of the "system-capabilities" data tree. This example assumes the "example-social" module defined in the Appendix A.1 is implemented.

```
   =============== NOTE: '\' line wrapping per RFC 8792 ================

   <system-capabilities
     xmlns="urn:ietf:params:xml:ns:yang:ietf-system-capabilities"
     xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores"
     xmlns:es="https://example.com/ns/example-social"
     xmlns:lpg="urn:ietf:params:xml:ns:yang:ietf-list-pagination">
     <datastore-capabilities>
       <datastore>ds:operational</datastore>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log</node-selector>
         <lpg:constrained>true</lpg:constrained>
       </per-node-capabilities>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log/es:timestamp</node-\
   selector>
         <lpg:indexed>true</lpg:indexed>
       </per-node-capabilities>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log/es:member-id</node-\
   selector>
         <lpg:indexed>true</lpg:indexed>
       </per-node-capabilities>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log/es:outcome</node-se\
   lector>
         <lpg:indexed>true</lpg:indexed>
       </per-node-capabilities>
     </datastore-capabilities>
   </system-capabilities>
```

4.3. YANG Module

This YANG module has normative references to [RFC7952], [RFC8342], [RFC9196], and [RFC9911].

```
   <CODE BEGINS> file "ietf-list-pagination@2026-07-06.yang"

   =============== NOTE: '\\' line wrapping per RFC 8792 ================

   module ietf-list-pagination {
     yang-version 1.1;
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-list-pagination";
     prefix lpg;

     import ietf-datastores {
       prefix ds;
```

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

  import ietf-yang-metadata {
    prefix md;
    reference
      "RFC 7952: Defining and Using Metadata with YANG";
  }

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

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 9911: Common YANG Data Types";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

     Author:   Kent Watsen
               <kent+ietf@watsen.net>

     Author:   Qin Wu
               <bill.wu@huawei.com>

     Author:   Per Andersson
               <per.ietf@ionio.se>

     Author:   Olof Hagsand
               <olof@hagsand.se>

     Author:   Hongwei Li
               <flycoolman@gmail.com>";

  description
    "This module is used by servers to 1) indicate they support
     pagination on 'list' and 'leaf-list' resources, 2) define a
```

```
        grouping for each list-pagination parameter, and 3) indicate
        which 'config false' lists have constrained 'where' and
        'sort-by' parameters and how they may be used, if at all.

        Copyright (c) 2026 IETF Trust and the persons identified
        as authors of the code. All rights reserved.

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

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

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
        'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
        'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
        are to be interpreted as described in BCP 14 (RFC 2119)
        (RFC 8174) when, and only when, they appear in all
        capitals, as shown here.";

    revision 2026-07-06 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: List Pagination for YANG-driven Protocols";
    }

    // Features

    feature sort-by {
      description
        "This feature indicates that the parameters 'locale' and
         'sort-by' are supported.";
    }

    feature where {
      description
        "This feature indicates that the parameter 'where' is
         supported";
    }
    // Annotations

    md:annotation remaining {
```

```
      type union {
        type uint32;
        type enumeration {
          enum "unknown" {
            description
              "Indicates that the number of remaining entries is
               unknown to the server in case, e.g., the server has
               determined that counting would be prohibitively
               expensive.";
          }
        }
      }
      description
        "This annotation contains the number of elements not included
         in the result set (a positive value) due to a 'limit' or
         'sublist-limit' operation.  If no elements were removed,
         this annotation MUST NOT appear.  The minimum value (0),
         which never occurs in normal operation, is reserved to
         represent 'unknown'.  The maximum value (2^32-1) is
         reserved to represent any value greater than or equal
         to 2^32-1 elements.";
      reference
        "RFC XXXX Section 3.1.7 The 'limit' Query Parameter";
    }

    md:annotation next {
      type string;
      description
        "This annotation contains the opaque encoded value of the next
         cursor in the pagination.";
      reference
        "RFC XXXX Section 3.1.6 The 'cursor' Query Parameter";
    }

    md:annotation previous {
      type string;
      description
        "This annotation contains the opaque encoded value of the
         previous cursor in the pagination.";
      reference
        "RFC XXXX Section 3.1.6 The 'cursor' Query Parameter";
    }

    md:annotation locale {
      if-feature "sort-by";
      type string;
      description
        "This annotation contains the locale used when sorting.
```

```
          The format is a free form string but SHOULD follow the
          language sub-tag format defined in RFC 5646.
          An example is 'sv_SE'.

          For further details see references:
          RFC 5646: Tags for identifying Languages
          RFC 6365: Technology Used in Internationalization in the
                    IETF";
      reference
        "RFC XXXX Section 3.1.3 The 'locale' Query Parameter";
    }

    // Identities

    identity list-pagination-error {
      description
        "Base identity for list-pagination errors.";
    }

    identity offset-out-of-range {
      base list-pagination-error;
      description
        "The 'offset' query parameter value is greater than the number
         of instances in the target list or leaf-list resource.";
    }

    identity cursor-not-found {
      base list-pagination-error;
      description
        "The 'cursor' query parameter value is unknown for the target
         list.";
    }

    identity locale-unavailable {
      if-feature "sort-by";
      base list-pagination-error;
      description
        "The 'locale' query parameter input is not a valid
         locale or the locale is not available on the system.";
    }

    // Groupings

    grouping pagination-parameters {
      description
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific pagination parameters.";
      container list-pagination {
```

```
        description
          "List pagination parameters.";
        leaf where {
          when "(../cursor = 'first')" {
            description
              "The 'where' query parameter is only allowed when
               initializing the cursor based pagination.";
          }
          if-feature "where";
          type union {
            type yang:xpath1.0;
            type enumeration {
              enum "unfiltered" {
                description
                  "Indicates that no entries are to be filtered
                   from the working result-set.";
              }
            }
          }
          default "unfiltered";
          description
            "The 'where' parameter specifies a boolean expression
             that result-set entries must match.

             If the XPath expression references a node identifier that
             does not exist in the schema, or is optional or
             conditional in the schema, the filter evaluates to false
             and nothing is filtered from the result-set.

             It is an error if the XPath expression references
             constrained 'config false' lists and leaf-lists, if the
             node identifier does not point to a node having the
             'indexed' leaf set to 'true' (see RFC XXXX).";
        }
        leaf locale {
          when "(../sort-by) and (../cursor = 'first')" {
            description
              "The 'locale' parameter may only be specified when the
               'sort-by' parameter is specified.

               The 'locale' query parameter is only allowed when
               initializing the cursor based pagination.";
          }
          type string;
          description
            "The 'locale' parameter indicates the locale which the
             entries in the working result-set should be collated.";
        }
```

```
        leaf sort-by {
          when "(../cursor = 'first')" {
            description
              "The 'sort-by' query parameter is only allowed when
               initializing the cursor based pagination.";
          }
          if-feature "sort-by";
          type union {
            type string {
              // An RFC 7950 'descendant-schema-nodeid'.
              pattern '([A-Za-z_][A-Za-z0-9_.-]*:)?'
                    + '[A-Za-z_][A-Za-z0-9_.-]*'
                    + '(/[A-Za-z_]([A-Za-z0-9_.-]*:)?'
                    + '[A-Za-z_]([A-Za-z0-9_.-]*))*';
            }
            type enumeration {
              enum "none" {
                description
                  "Indicates that the list or leaf-list's default
                   order is to be used, per the YANG 'ordered-by'
                   statement.";
              }
            }
          }
          default "none";
          description
            "The 'sort-by' parameter indicates the node in the
             working result-set (i.e., after the 'where' parameter
             has been applied) that entries should be sorted by.

             Sorts are in ascending order (e.g., '1' before '9',
             'a' before 'z', etc.).  Missing values are sorted to
             the end (e.g., after all nodes having values).";
        }
        leaf direction {
          when "(../cursor = 'first')" {
            description
              "The 'direction' query parameter is only allowed when
               initializing the cursor based pagination.";
          }
          type enumeration {
            enum forwards {
              description
                 "Indicates that entries should be traversed from
                  the first to last item in the working result set.";
            }
            enum backwards {
              description
```

```
                "Indicates that entries should be traversed from
                 the last to first item in the working result set.";
            }
          }
          default "forwards";
          description
            "The 'direction' parameter indicates how the entries in the
             working result-set (i.e., after the 'sort-by' parameter
             has been applied) should be traversed.";
        }
        choice navigation-type {
          case cursor {
            leaf cursor {
              type union {
                type string;
                type enumeration {
                  enum "first" {
                    description
                      "Indicates to start at the first item in the
                       result-set.";
                  }
                }
              }
              default "first";
              description
                "The 'cursor' parameter indicates where to start the
                 working result-set (i.e. after the 'direction' parame\
\ter
                 has been applied), the elements before the cursor are
                 skipped over when preparing the response. Furthermore\
\ the
                 result-set is annotated with attributes for the next \
\and
                 previous cursors following a result-set constrained w\
\ith
                 the 'limit' query parameter.";
            }
          }
          case offset {
            leaf offset {
              type uint32;
              default 0;
              description
                "The 'offset' parameter indicates the number of entries
                 in the working result-set (i.e., after the 'direction'
                 parameter has been applied) that should be skipped ov\
\er
                 when preparing the response.";
```

```
            }
          }
          description
            "Indicates is navigation is by ‘offset‘ or ‘cursor‘.  Whil\
   \st
             both options have a default value indicating to start at \
   \the
             beginning of the result set, the ’offset’ option is defau\
   \lt
             so that there is no expectation that the cursor’s next/pr\
   \ev
             values will be returned.";
        }
        leaf limit {
          type union {
            type uint32 {
              range "1..max";
            }
            type enumeration {
              enum "unbounded" {
                description
                  "Indicates that the number of entries that may be
                   returned is unbounded.";
              }
            }
          }
          default "unbounded";
          description
            "The ’limit’ parameter limits the number of entries
             returned from the working result-set (i.e., after the
             ’offset’ parameter has been applied).

             Any result-set that is limited includes, somewhere in its
             encoding, the metadata value ’remaining’ to indicate the
             number entries not included in the result set.";
        }
        leaf sublist-limit {
          type union {
            type uint32 {
              range "1..max";
            }
            type enumeration {
              enum "unbounded" {
                description
                  "Indicates that the number of entries that may be
                   returned is unbounded.";
              }
            }
```

```
          }
          default "unbounded";
          description
            "The 'sublist-limit' parameter limits the number of entries
             for descendant lists and leaf-lists.

             Any result-set that is limited includes, somewhere in
             its encoding, the metadata value 'remaining' to indicate
             the number entries not included in the result set.";
        }
      }
    }

    // Protocol-accessible nodes

    augment
      "/sysc:system-capabilities/sysc:datastore-capabilities"
      + "/sysc:per-node-capabilities" {

      // Ensure the following nodes are only used for the
      // <operational> datastore.
      when "/sysc:system-capabilities/sysc:datastore-capabilities"
           + "/sysc:datastore = 'ds:operational'";

      description
        "Defines some leafs that MAY be used by the server to
         describe constraints imposed of the 'where' filters and
         'sort-by' parameters used in list pagination queries.";

      leaf constrained {
        type boolean;
        default false;
        description
          "Indicates that 'where' filters and 'sort-by' parameters
           on the targeted 'config false' list node are constrained.
           If a list is not 'constrained', then full XPath 1.0
           expressions may be used in 'where' filters and all node
           identifiers are usable by 'sort-by'.

           Setting this to true will make the list constrained.

           By default this is false, i.e. a list is not constrained.";
      }
      leaf indexed {
        type boolean;
        default false;
        description
          "Indicates that the targeted descendant node of a
```

```
            'constrained' list (see the 'constrained' leaf) may be
            used in 'where' filters and/or 'sort-by' parameters.
            If a descendant node of a 'constrained' list is not
            'indexed', then it MUST NOT be used in 'where' filters
            or 'sort-by' parameters.

            Setting this to true will allow a constrained list to be
            used in 'where' filters and/or 'sort-by' parameters.

            By default this is false, i.e. a list constrained is not
            allowed for 'where' filters or 'sort-by' parameters.";
        }
        leaf cursor-supported {
          type boolean;
          default false;
          description
            "Indicates that the targeted list node supports the
             'cursor' parameter.

             Setting this to true will enable the 'cursor' query
             parameter for a 'config false' list.

             By default this is false, i.e. the 'cursor' query parameter
             is disabled for 'config false' lists.";
        }
      }
    }
```

<CODE ENDS>

5. IANA Considerations

5.1. The "IETF XML" Registry

This document registers one URI in the "ns" subregistry of the IETF XML Registry [RFC3688] maintained at https://www.iana.org/assignments/xml-registry/xml-registry.xhtml#ns. Following the format in [RFC3688], the following registration is requested:

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

5.2. The "YANG Module Names" Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020] maintained at https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml. Following the format defined in [RFC6020], the below registration is requested:

```
name: ietf-list-pagination
namespace: urn:ietf:params:xml:ns:yang:ietf-list-pagination
prefix: lpg
RFC: XXXX
```

6. Operational Considerations

This section is modeled after Section 3 of [I-D.ietf-opsawg-rfc5706bis].

When using the groupings defined in this document, it is possible to refine the leafs and add default values. Such default values will be used as default settings for requests.

The operational considerations regarding list-pagination are multifold: The protocol and transport and also for the underlying database. The performance and even capabilities for efficient indexing and pagination is something to be considered in deployments. The system-capabilities subtree is augmented with capabilities and constraints for list-pagination capabilities supported by the system.

It is easy to create queries that would create exhaust resources and thus create denial-of-service scenarios. Huge data sets might need to be constrained by disallowing indexing. This is something that deployments need to consider. Again, it is possible to constrain list-pagination on subtrees with the system-capabilities subtree.

7. Security Considerations

7.1. Considerations for the "ietf-list-pagination" YANG Module

This section follows the template defined in Section 3.7.1 of [RFC8407].

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

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

All protocol-accessible data nodes in this module are read-only and cannot be modified. Access control may be configured to avoid exposing any read-only data that is defined by the augmenting module documentation as being security sensitive.

Since this module also defines groupings, these considerations are primarily for the designers of other modules that use these groupings.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

This module does not define any RPCs or actions or notifications, and thus the security consideration for such is not provided here.

8. References

8.1. Normative References

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

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

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

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

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

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

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

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

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

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

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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/info/rfc9196>.

[RFC9911] SchÃ¶nwÃ¤lder, J., Ed., "Common YANG Data Types", RFC 9911, DOI 10.17487/RFC9911, December 2025, <https://www.rfc-editor.org/info/rfc9911>.

[I-D.ietf-opsawg-rfc5706bis]
Claise, B., Clarke, J., Farrel, A., Barguil, S., Pignataro, C., and R. Chen, "Guidelines for Considering Operations and Management in IETF Specifications", Work in Progress, Internet-Draft, draft-ietf-opsawg-rfc5706bis-05, 26 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-rfc5706bis-05>.

8.2. Informative References

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

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

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

[I-D.ietf-netconf-list-pagination-nc]
Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "NETCONF Extensions to Support List Pagination", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-nc-11, 1 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-nc-11>.

[I-D.ietf-netconf-list-pagination-rc]
Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "RESTCONF Extensions to Support List Pagination", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-rc-10, 12 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-rc-10>.

[I-D.ietf-netconf-restconf-collection]
Bierman, A., BjÃ¶rklund, M., and K. Watsen, "RESTCONF Collection Resource", Work in Progress, Internet-Draft,

draft-ietf-netconf-restconf-collection-00, 30 January 2015, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-restconf-collection-00>.

[I-D.zheng-netconf-fragmentation]
Zheng, G. and Z. Wang, "A NETCONF Extension for Data Fragmentation", Work in Progress, Internet-Draft, draft-zheng-netconf-fragmentation-02, 26 June 2018, <https://datatracker.ietf.org/doc/html/draft-zheng-netconf-fragmentation-02>.

Appendix A. Vector Tests

This normative appendix section illustrates every notable edge condition conceived during this document's production.

Test inputs and outputs are provided in a manner that is both generic and concise.

Management protocol specific documents need only reproduce as many of these tests as necessary to convey peculiarities presented by the protocol.

Implementations are RECOMMENDED to implement the tests presented in this document, in addition to any tests that may be presented in protocol specific documents.

A.1. Example YANG Module

The vector tests assume the "example-social" YANG module defined in this section.

This module has been specially crafted to cover every notable edge condition, especially with regards to the types of the data nodes.

Following is the tree diagram [RFC8340] for the "example-social" module:

```
module: example-social
  +--rw members
  |  +--rw member* [member-id]
  |     +--rw member-id           string
  |     +--rw email-address       inet:email-address
  |     +--rw password            ianach:crypt-hash
  |     +--rw avatar?             binary
  |     +--rw tagline?            string
  |     +--rw privacy-settings
  |     |  +--rw hide-network?      boolean
  |     |  +--rw post-visibility?   enumeration
  |     +--rw following*          -> /members/member/member-id
  |     +--rw posts
  |     |  +--rw post* [timestamp]
  |     |     +--rw timestamp    yang:date-and-time
  |     |     +--rw title?       string
  |     |     +--rw body         string
  |     +--rw favorites
  |     |  +--rw uint8-numbers*       uint8
  |     |  +--rw uint64-numbers*      uint64
  |     |  +--rw int8-numbers*        int8
  |     |  +--rw int64-numbers*       int64
  |     |  +--rw decimal64-numbers*   decimal64
  |     |  +--rw bits*                bits
  |     +--ro stats
  |        +--ro joined              yang:date-and-time
  |        +--ro membership-level    enumeration
  |        +--ro last-activity?      yang:date-and-time
  +--ro audit-logs
     +--ro audit-log* []
        +--ro timestamp    yang:date-and-time
        +--ro member-id    string
        +--ro source-ip    inet:ip-address
        +--ro request      string
        +--ro outcome      boolean
```

Following is the YANG [RFC7950] for the "example-social" module:

```
module example-social {
  yang-version 1.1;
  namespace "https://example.com/ns/example-social";
  prefix es;

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

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

import iana-crypt-hash {
  prefix ianach;
  reference
    "RFC 7317: A YANG Data Model for System Management";
}

organization "Example, Inc.";
contact      "support@example.com";
description  "Example Social Data Model.";

revision YYYY-MM-DD {
  description
    "Initial version.";
  reference
    "RFC XXXX: Example social module.";
}

container members {
  description
    "Container for list of members.";
  list member {
    key "member-id";
    description
      "List of members.";

    leaf member-id {
      type string {
        length "1..80";
        pattern '.*[\n].*' {
         modifier invert-match;
        }
      }
      description
        "The member's identifier.";
    }

    leaf email-address {
      type inet:email-address;
      mandatory true;
      description
        "The member's email address.";
    }
```

```
        leaf password {
          type ianach:crypt-hash;
          mandatory true;
          description
            "The member's hashed-password.";
        }

        leaf avatar {
          type binary;
          description
            "An binary image file.";
        }

        leaf tagline {
          type string {
            length "1..80";
            pattern '.*[\n].*' {
              modifier invert-match;
            }
          }
          description
            "The member's tagline.";
        }

        container privacy-settings {
          leaf hide-network {
            type boolean;
            description
              "Hide who you follow and who follows you.";
          }
          leaf post-visibility {
            type enumeration {
              enum public {
                description
                  "Posts are public.";
              }
              enum unlisted {
                description
                  "Posts are unlisted, though visible to all.";
              }
              enum followers-only {
                description
                  "Posts only visible to followers.";
              }
            }
            default public;
            description
              "The post privacy setting.";
```

```
      }
      description
        "Preferences for the member.";
    }

    leaf-list following {
      type leafref {
        path "/members/member/member-id";
      }
      description
        "Other members this member is following.";
    }

    container posts {
      description
        "The member's posts.";
      list post {
        key timestamp;
        leaf timestamp {
          type yang:date-and-time;
          description
            "The timestamp for the member's post.";
        }
        leaf title {
          type string {
            length "1..80";
            pattern '.*[\n].*' {
              modifier invert-match;
            }
          }
          description
            "A one-line title.";
        }
        leaf body {
          type string;
          mandatory true;
          description
            "The body of the post.";
        }
        description
          "A list of posts.";
      }
    }

    container favorites {
      description
        "The member's favorites.";
      leaf-list uint8-numbers {
```

```
      type uint8;
      ordered-by user;
      description
        "The member's favorite uint8 numbers.";
    }
    leaf-list uint64-numbers {
      type uint64;
      ordered-by user;
      description
        "The member's favorite uint64 numbers.";
    }
    leaf-list int8-numbers {
      type int8;
      ordered-by user;
      description
        "The member's favorite int8 numbers.";
    }
    leaf-list int64-numbers {
      type int64;
      ordered-by user;
      description
        "The member's favorite int64 numbers.";
    }
    leaf-list decimal64-numbers {
      type decimal64 {
        fraction-digits 5;
      }
      ordered-by user;
      description
        "The member's favorite decimal64 numbers.";
    }
    leaf-list bits {
      type bits {
        bit zero {
          position 0;
          description "zero";
        }
        bit one {
          position 1;
          description "one";
        }
        bit two {
          position 2;
          description "two";
        }
      }
      ordered-by user;
      description
```

```
            "The member's favorite bits.";
        }
      }

      container stats {
        config false;
        description
          "Operational state members values.";
        leaf joined {
          type yang:date-and-time;
          mandatory true;
          description
            "Timestamp when member joined.";
        }
        leaf membership-level {
          type enumeration {
            enum admin {
              description
                "Site administrator.";
            }
            enum standard {
              description
                "Standard membership level.";
            }
            enum pro {
              description
                "Professional membership level.";
            }
          }
          mandatory true;
          description
            "The membership level for this member.";
        }
        leaf last-activity {
          type yang:date-and-time;
          description
            "Timestamp of member's last activity.";
        }
      }
    }
  }

  container audit-logs {
    config false;
    description
      "Audit log configuration";
    list audit-log {
      description
```

```
          "List of audit logs.";
        leaf timestamp {
          type yang:date-and-time;
          mandatory true;
          description
            "The timestamp for the event.";
        }
        leaf member-id {
          type string;
          mandatory true;
          description
            "The 'member-id' of the member.";
        }
        leaf source-ip {
          type inet:ip-address;
          mandatory true;
          description
            "The apparent IP address the member used.";
        }
        leaf request {
          type string;
          mandatory true;
          description
            "The member's request.";
        }
        leaf outcome {
          type boolean;
          mandatory true;
          description
            "Indicate if request was permitted.";
        }
      }
    }
  }
```

A.2. Example Data Set

The examples assume the server's operational state as follows.

The data is provided in JSON only for convenience and, in particular, has no bearing on the "generic" nature of the tests themselves.

```
{
  "example-social:members": {
    "member": [
      {
        "member-id": "bob",
        "email-address": "bob@example.com",
```

```
          "password": "$0$1543",
          "avatar": "BASE64VALUE=",
          "tagline": "Here and now, like never before.",
          "posts": {
            "post": [
              {
                "timestamp": "2020-08-14T03:32:25Z",
                "body": "Just got in."
              },
              {
                "timestamp": "2020-08-14T03:33:55Z",
                "body": "What's new?"
              },
              {
                "timestamp": "2020-08-14T03:34:30Z",
                "body": "I'm bored..."
              }
            ]
          },
          "favorites": {
            "decimal64-numbers": ["3.14159", "2.71828"]
          },
          "stats": {
            "joined": "2020-08-14T03:30:00Z",
            "membership-level": "standard",
            "last-activity": "2020-08-14T03:34:30Z"
          }
        },
        {
          "member-id": "eric",
          "email-address": "eric@example.com",
          "password": "$0$1543",
          "avatar": "BASE64VALUE=",
          "tagline": "Go to bed with dreams; wake up with a purpose.",
          "following": ["alice"],
          "posts": {
            "post": [
              {
                "timestamp": "2020-09-17T18:02:04Z",
                "title": "Son, brother, husband, father",
                "body": "What's your story?"
              }
            ]
          },
          "favorites": {
            "bits": ["two", "one", "zero"]
          },
          "stats": {
```

```
            "joined": "2020-09-17T19:38:32Z",
            "membership-level": "pro",
            "last-activity": "2020-09-17T18:02:04Z"
          }
        },
        {
          "member-id": "alice",
          "email-address": "alice@example.com",
          "password": "$0$1543",
          "avatar": "BASE64VALUE=",
          "tagline": "Every day is a new day",
          "privacy-settings": {
            "hide-network": false,
            "post-visibility": "public"
          },
          "following": ["bob", "eric", "lin"],
          "posts": {
            "post": [
              {
                "timestamp": "2020-07-08T13:12:45Z",
                "title": "My first post",
                "body": "Hiya all!"
              },
              {
                "timestamp": "2020-07-09T01:32:23Z",
                "title": "Sleepy...",
                "body": "Catch y'all tomorrow."
              }
            ]
          },
          "favorites": {
            "uint8-numbers": [17, 13, 11, 7, 5, 3],
            "int8-numbers": [-5, -3, -1, 1, 3, 5]
          },
          "stats": {
            "joined": "2020-07-08T12:38:32Z",
            "membership-level": "admin",
            "last-activity": "2021-04-01T02:51:11Z"
          }
        },
        {
          "member-id": "lin",
          "email-address": "lin@users.example.net",
          "password": "$0$1543",
          "privacy-settings": {
            "hide-network": true,
            "post-visibility": "followers-only"
          },
```

```
          "following": ["joe", "eric", "alice"],
          "stats": {
            "joined": "2020-07-09T12:38:32Z",
            "membership-level": "standard",
            "last-activity": "2021-04-01T02:51:11Z"
          }
        },
        {
          "member-id": "joe",
          "email-address": "joe@example.com",
          "password": "$0$1543",
          "avatar": "BASE64VALUE=",
          "tagline": "Greatness is measured by courage and heart.",
          "privacy-settings": {
            "post-visibility": "unlisted"
          },
          "following": ["bob"],
          "posts": {
            "post": [
              {
                "timestamp": "2020-10-17T18:02:04Z",
                "body": "What's your status?"
              }
            ]
          },
          "stats": {
            "joined": "2020-10-08T12:38:32Z",
            "membership-level": "pro",
            "last-activity": "2021-04-01T02:51:11Z"
          }
        },
        {
          "member-id": "Ã¥sa",
          "email-address": "asa@users.example.net",
          "password": "$0$1543",
          "avatar": "BASE64VALUE=",
          "privacy-settings": {
            "post-visibility": "unlisted"
          },
          "following": ["alice", "bob"],
          "stats": {
            "joined": "2022-02-19T13:12:00Z",
            "membership-level": "standard",
            "last-activity": "2022-04-19T13:12:59Z"
          }
        }
      ]
    },
```

```
    "example-social:audit-logs": {
      "audit-log": [
        {
          "timestamp": "2020-10-11T06:47:59Z",
          "member-id": "alice",
          "source-ip": "192.168.0.92",
          "request": "POST /groups/group/2043",
          "outcome": true
        },
        {
          "timestamp": "2020-11-01T15:22:01Z",
          "member-id": "bob",
          "source-ip": "192.168.2.16",
          "request": "POST /groups/group/123",
          "outcome": false
        },
        {
          "timestamp": "2020-12-12T21:00:28Z",
          "member-id": "eric",
          "source-ip": "192.168.254.1",
          "request": "POST /groups/group/10",
          "outcome": true
        },
        {
          "timestamp": "2021-01-03T06:47:59Z",
          "member-id": "alice",
          "source-ip": "192.168.0.92",
          "request": "POST /groups/group/333",
          "outcome": true
        },
        {
          "timestamp": "2021-01-21T10:00:00Z",
          "member-id": "bob",
          "source-ip": "192.168.2.16",
          "request": "POST /groups/group/42",
          "outcome": true
        },
        {
          "timestamp": "2020-02-07T09:06:21Z",
          "member-id": "alice",
          "source-ip": "192.168.0.92",
          "request": "POST /groups/group/1202",
          "outcome": true
        },
        {
          "timestamp": "2020-02-28T02:48:11Z",
          "member-id": "bob",
          "source-ip": "192.168.2.16",
```

```
          "request": "POST /groups/group/345",
          "outcome": true
        }
      ]
    }
  }
```

A.3. Example Queries

The following sections are presented in reverse query-parameters processing order. Starting with the simplest (limit) and ending with the most complex (where).

All the vector tests are presented in a protocol-independent manner. JSON is used only for its conciseness.

The "members" list in the example data set is "ordered-by system" and the queries without explicit "sort-by" below return them in this given order. Note that the order of the result-set without explicit "sort-by" is implementation specific for "ordered-by system" lists.

> Note that the user "Ã¥sa" is only included in Appendix A.3.7. This to isolate the parameter in the example query and its behavior.

A.3.1. The "limit" Parameter

Noting that "limit" must be a positive number, the edge condition values are '1', '2', num-elements-1, num-elements, and num-elements+1.

> Any value greater than or equal to num-elements results the entire result set, same as when "limit" is unspecified.

These vector tests assume the target "/example-social:members/member=alice/favorites/uint8-numbers", which has six values, thus the edge condition "limit" values are: '1', '2', '5', '6', and '7'.

A.3.1.1. limit=1

REQUEST

```
   Target: /example-social:members/member=alice/favorites/uint8-numbers
     Pagination Parameters:
       Where:     -
       Sort-by:   -
       Direction: -
       Offset:    -
       Limit:     1

   RESPONSE

   {
     "example-social:uint8-numbers": [17],
     "@example-social:uint8-numbers": [
        {
           "ietf-list-pagination:remaining": 5
        }
      ]
   }
```

A.3.1.2. limit=2

```
   REQUEST

   Target: /example-social:members/member=alice/favorites/uint8-numbers
     Pagination Parameters:
       Where:     -
       Sort-by:   -
       Direction: -
       Offset:    -
       Limit:     2

   RESPONSE

   {
     "example-social:uint8-numbers": [17, 13],
     "@example-social:uint8-numbers": [
        {
           "ietf-list-pagination:remaining": 4
        }
      ]
   }
```

A.3.1.3. limit=5

```
   REQUEST
```

```
   Target: /example-social:members/member=alice/favorites/uint8-numbers
     Pagination Parameters:
       Where:     -
       Sort-by:   -
       Direction: -
       Offset:    -
       Limit:     5

   RESPONSE

   {
     "example-social:uint8-numbers": [17, 13, 11, 7, 5],
     "@example-social:uint8-numbers": [
        {
           "ietf-list-pagination:remaining": 1
        }
      ]
   }
```

A.3.1.4. limit=6

```
   REQUEST

   Target: /example-social:members/member=alice/favorites/uint8-numbers
     Pagination Parameters:
       Where:     -
       Sort-by:   -
       Direction: -
       Offset:    -
       Limit:     6

   RESPONSE

   {
     "example-social:uint8-numbers": [17, 13, 11, 7, 5, 3]
   }
```

A.3.1.5. limit=7

```
   REQUEST

   Target: /example-social:members/member=alice/favorites/uint8-numbers
     Pagination Parameters:
       Where:     -
       Sort-by:   -
       Direction: -
       Offset:    -
       Limit:     7
```

RESPONSE

```
{
  "example-social:uint8-numbers": [17, 13, 11, 7, 5, 3]
}
```

A.3.2. The "offset" Parameter

Noting that "offset" must be an unsigned number less than or equal to the num-elements, the edge condition values are '0', '1', '2', num-elements-1, num-elements, and num-elements+1.

These vector tests again assume the target "/example-social:members/member=alice/favorites/uint8-numbers", which has six values, thus the edge condition "limit" values are: '0', '1', '2', '5', '6', and '7'.

A.3.2.1. offset=0

REQUEST

```
Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    0
    Limit:     -
```

RESPONSE

```
{
  "example-social:uint8-numbers": [17, 13, 11, 7, 5, 3]
}
```

A.3.2.2. offset=1

REQUEST

```
Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    1
    Limit:     -
```

RESPONSE

```
{
  "example-social:uint8-numbers": [13, 11, 7, 5, 3]
}
```

A.3.2.3. offset=2

```
REQUEST

Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    2
    Limit:     -

RESPONSE

{
  "example-social:uint8-numbers": [11, 7, 5, 3]
}
```

A.3.2.4. offset=5

```
REQUEST

Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    5
    Limit:     -

RESPONSE

{
  "example-social:uint8-numbers": [3]
}
```

A.3.2.5. offset=6

```
REQUEST
```

```
Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    6
    Limit:     -

RESPONSE

{
  "example-social:uint8-numbers": []
}
```

A.3.2.6. offset=7

```
REQUEST

Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    7
    Limit:     -

RESPONSE

error-type: application
error-tag: invalid-value
error-app-tag: ietf-list-pagination:offset-out-of-range
```

A.3.3. The "cursor" Parameter

Noting that "cursor" is an opaque encoded value represented by a string, which addresses an element in a list.

> The default value is empty, which is the same as supplying the cursor value for the first element in the list.

These vector tests assume the target "/example-social:members/member" which has five members.

> Note that response has added attributes describing the result set and position in pagination.

A.3.3.1. Cursor using the Default Value

REQUEST

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     2
    Cursor:    -
```

RESPONSE

```
{
  "example-social:member": [
    {
      "member-id": "bob",
      "email-address": "bob@example.com",
      "password": "$0$1543",
      "avatar": "BASE64VALUE=",
      "tagline": "Here and now, like never before.",
      "posts": {
        "post": [
          {
            "timestamp": "2020-08-14T03:32:25Z",
            "body": "Just got in."
          },
          {
            "timestamp": "2020-08-14T03:33:55Z",
            "body": "What's new?"
          },
          {
            "timestamp": "2020-08-14T03:34:30Z",
            "body": "I'm bored..."
          }
        ]
      },
      "favorites": {
        "decimal64-numbers": ["3.14159", "2.71828"]
      },
      "stats": {
        "joined": "2020-08-14T03:30:00Z",
        "membership-level": "standard",
        "last-activity": "2020-08-14T03:34:30Z"
      }
    },
```

```
      {
        "member-id": "eric",
        "email-address": "eric@example.com",
        "password": "$0$1543",
        "avatar": "BASE64VALUE=",
        "tagline": "Go to bed with dreams; wake up with a purpose.",
        "following": ["alice"],
        "posts": {
          "post": [
            {
              "timestamp": "2020-09-17T18:02:04Z",
              "title": "Son, brother, husband, father",
              "body": "What's your story?"
            }
          ]
        },
        "favorites": {
          "bits": ["two", "one", "zero"]
        },
        "stats": {
          "joined": "2020-09-17T19:38:32Z",
          "membership-level": "pro",
          "last-activity": "2020-09-17T18:02:04Z"
        }
      }
    ],
    "@example-social:member": [
      {
        "ietf-list-pagination:remaining": 3,
        "ietf-list-pagination:previous": "",
        "ietf-list-pagination:next": "YWxpY2U="
      }
    ]
  }
```

A.3.3.2. Cursor Using the "next" Value

```
  REQUEST

  Target: /example-social:members/member
    Pagination Parameters:
      Where:     -
      Sort-by:   -
      Direction: -
      Offset:    -
      Limit:     2
      Cursor:    YWxpY2U=
```

RESPONSE


```
{
  "example-social:member": [
    {
      "member-id": "alice",
      "email-address": "alice@example.com",
      "password": "$0$1543",
      "avatar": "BASE64VALUE=",
      "tagline": "Every day is a new day",
      "privacy-settings": {
        "hide-network": false,
        "post-visibility": "public"
      },
      "following": ["bob", "eric", "lin"],
      "posts": {
        "post": [
          {
            "timestamp": "2020-07-08T13:12:45Z",
            "title": "My first post",
            "body": "Hiya all!"
          },
          {
            "timestamp": "2020-07-09T01:32:23Z",
            "title": "Sleepy...",
            "body": "Catch y'all tomorrow."
          }
        ]
      },
      "favorites": {
        "uint8-numbers": [17, 13, 11, 7, 5, 3],
        "int8-numbers": [-5, -3, -1, 1, 3, 5]
      },
      "stats": {
        "joined": "2020-07-08T12:38:32Z",
        "membership-level": "admin",
        "last-activity": "2021-04-01T02:51:11Z"
      }
    },
    {
      "member-id": "lin",
      "email-address": "lin@example.com",
      "password": "$0$1543",
      "privacy-settings": {
        "hide-network": true,
        "post-visibility": "followers-only"
      },
      "following": ["joe", "eric", "alice"],
```

```
        "stats": {
          "joined": "2020-07-09T12:38:32Z",
          "membership-level": "standard",
          "last-activity": "2021-04-01T02:51:11Z"
        }
      }
    ],
    "@example-social:member": [
      {
        "ietf-list-pagination:remaining": 1,
        "ietf-list-pagination:previous": "ZXJpYw==",
        "ietf-list-pagination:next": "am9l"
      }
    ]
  }
```

A.3.3.3. Cursor Using Another "next" Value

```
REQUEST

Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     2
    Cursor:    am9l

RESPONSE
```

```
{
  "example-social:member": [
    {
      "member-id": "joe",
      "email-address": "joe@example.com",
      "password": "$0$1543",
      "avatar": "BASE64VALUE=",
      "tagline": "Greatness is measured by courage and heart.",
      "privacy-settings": {
        "post-visibility": "unlisted"
      },
      "following": ["bob"],
      "posts": {
        "post": [
          {
            "timestamp": "2020-10-17T18:02:04Z",
            "body": "What's your status?"
          }
        ]
      },
      "stats": {
        "joined": "2020-10-08T12:38:32Z",
        "membership-level": "pro",
        "last-activity": "2021-04-01T02:51:11Z"
      }
    }
  ],
  "@example-social:member": [
    {
      "ietf-list-pagination:remaining": 0,
      "ietf-list-pagination:previous": "bGlu",
      "ietf-list-pagination:next": ""
    }
  ]
}
```


A.3.3.4. Cursor Using an Invalid Value

REQUEST

The cursor used does not exist in the datastore.

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     -
    Cursor:    <invalid-value>

RESPONSE

error-type: application
error-tag: invalid-value
error-app-tag: ietf-list-pagination:cursor-not-found
```

A.3.4. The "direction" Parameter

Noting that "direction" is an enumeration with two values, the edge condition values are each defined enumeration.

> The value "forwards" is sometimes known as the "default" value, as it produces the same result set as when "direction" is unspecified.

These vector tests again assume the target "/example-social:members/member=alice/favorites/uint8-numbers". The number of elements is relevant to the edge condition values.

> It is notable that "uint8-numbers" is an "ordered-by" user leaf-list. Traversals are over the user-specified order, not the numerically-sorted order, which is what the "sort-by" parameter addresses. If this were an "ordered-by system" leaf-list, then the traversals would be over the system-specified order, again not a numerically-sorted order.

A.3.4.1. direction=forwards

```
REQUEST

Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: forwards
    Offset:    -
    Limit:     -

RESPONSE
```

```
{
  "example-social:uint8-numbers": [17, 13, 11, 7, 5, 3]
}
```

A.3.4.2. direction=backwards

REQUEST

```
Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: backwards
    Offset:    -
    Limit:     -
```

RESPONSE

```
{
  "example-social:uint8-numbers": [3, 5, 7, 11, 13, 17]
}
```

A.3.5. The "sort-by" Parameter

Noting that the "sort-by" parameter is a node identifier, there is not so much "edge conditions" as there are "interesting conditions". This section provides examples for some interesting conditions.

A.3.5.1. the target node's type

The section provides three examples, one for a "leaf-list" and two for a "list", with one using a direct descendant and the other using an indirect descendant.

A.3.5.1.1. type is a "leaf-list"

This example illustrates when the target node's type is a "leaf-list". Note that a single period (i.e., '.') is used to represent the nodes to be sorted.

This test again uses the target "/example-social:members/member=alice/favorites/uint8-numbers", which is a leaf-list.

REQUEST

```
Target: /example-social:members/member=alice/favorites/uint8-numbers
  Pagination Parameters:
    Where:     -
    Sort-by:   .
    Direction: -
    Offset:    -
    Limit:     -

RESPONSE


{
  "example-social:uint8-numbers": [3, 5, 7, 11, 13, 17]
}
```

A.3.5.1.2. type is a "list" and sort-by node is a direct descendant

This example illustrates when the target node's type is a "list" and a direct descendant is the "sort-by" node.

This vector test uses the target "/example-social:members/member", which is a "list", and the sort-by descendant node "member-id", which is the "key" for the list.

REQUEST

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   member-id
    Direction: -
    Offset:    -
    Limit:     -
```

RESPONSE

> To make the example more understandable, an ellipse (i.e., "...") is used to represent a missing subtree of data.

```
{
  "example-social:member": [
    {
      "member-id": "alice",
      ...
    },
    {
      "member-id": "bob",
      ...
    },
    {
      "member-id": "eric",
      ...
    },
    {
      "member-id": "joe",
      ...
    },
    {
      "member-id": "lin",
      ...
    }
  ]
}
```

A.3.5.1.3. type is a "list" and sort-by node is an indirect descendant

This example illustrates when the target node's type is a "list" and an indirect descendant is the "sort-by" node.

This vector test uses the target "/example-social:members/member", which is a "list", and the sort-by descendant node "stats/joined", which is a "config false" descendant leaf. Due to "joined" being a "config false" node, this request would have to target the "member" node in the <operational> datastore.

REQUEST

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   stats/joined
    Direction: -
    Offset:    -
    Limit:     -
```

RESPONSE

> To make the example more understandable, an elipse (i.e., "...") is used to represent a missing subtree of data.

```
{
  "example-social:member": [
    {
      "member-id": "alice",
      ...
    },
    {
      "member-id": "lin",
      ...
    },
    {
      "member-id": "bob",
      ...
    },
    {
      "member-id": "eric",
      ...
    },
    {
      "member-id": "joe",
      ...
    }
  ]
}
```

A.3.6. The "where" Parameter

The "where" is an extended XPath 1.0 expression, there are numerous edge conditions to consider, e.g., the types of the nodes that are targeted by the expression.

A.3.6.1. match of leaf-list's values

This example selects the uint8-numbers greater than 7 in the member alice's favorites.

REQUEST

```
Target: /example-social:members/member=alice/favorites
  Pagination Parameters:
    Where:     uint8-numbers[. > 7]
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     -
```

RESPONSE

```
{
  "example-social:uint8-numbers": [17, 13, 11],
}
```

A.3.6.2. match on descendant string containing a substring

This example selects members that have an email address containing "@example.com".

REQUEST

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     .[contains (email-address,'@example.com')]
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     -
```

RESPONSE

> To make the example more understandable, an elipse (i.e., "...") is used to represent a missing subtree of data.

```
{
  "example-social:member": [
    {
      "member-id": "bob",
      ...
    },
    {
      "member-id": "eric",
      ...
    },
    {
      "member-id": "alice",
      ...
    },
    {
      "member-id": "joe",
      ...
    }
  ]
}
```

A.3.6.3. match on decendent timestamp starting with a substring

This example selects members that have a posting whose timestamp begins with the string "2020".

REQUEST

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     posts/post[starts-with(timestamp,'2020')]
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     -
```

RESPONSE

> To make the example more understandable, an elipse (i.e., "...") is used to represent a missing subtree of data.

```
{
  "example-social:member": [
    {
      "member-id": "bob",
      ...
    },
    {
      "member-id": "eric",
      ...
    },
    {
      "member-id": "alice",
      ...
    },
    {
      "member-id": "joe",
      ...
    }
  ]
}
```

A.3.7. The "locale" Parameter

The "locale" parameter may be used on any target node.

If this parameter is omitted, there is no default value and it is up to the server to choose a locale. This locale is then reported in the result-set as the "locale" metadata value.

Note that for ordered-by user lists and leaf-lists "locale" is not relevant, since the order is set by the user. For ordered-by system lists and leaf-lists, the server MAY report "locale" if the order that the server has chosen follows a valid locale,

If "locale" is used on an ordered-by user list, error-type "application" and error-tag "invalid-value" is returned.

If an ordered-by system target is not ordered according to any locale, the server omits the locale from the response.

REQUEST

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   member-id
    Direction: -
    Offset:    -
    Limit:     -
    Locale:    sv_SE
```

RESPONSE

```
{
  "example-social:member": [
    {
      "member-id": "alice",
      ...
    },
    {
      "member-id": "bob",
      ...
    },
    {
      "member-id": "eric",
      ...
    },
    {
      "member-id": "joe",
      ...
    },
    {
      "member-id": "lin",
      ...
    },
    {
      "member-id": "Ã¥sa",
      ...
    }
  ],
  "@example-social:member": [
    {
      "ietf-list-pagination:locale": "sv_SE"
    }
  ]
}
```

```
REQUEST

Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   member-id
    Direction: -
    Offset:    -
    Limit:     -
    Locale:    en_US

RESPONSE
```

```
{
  "example-social:member": [
    {
      "member-id": "alice",
      ...
    },
    {
      "member-id": "Ã¥sa",
      ...
    },
    {
      "member-id": "bob",
      ...
    },
    {
      "member-id": "eric",
      ...
    },
    {
      "member-id": "joe",
      ...
    },
    {
      "member-id": "lin",
      ...
    }
  ],
  "@example-social:member": [
    {
      "ietf-list-pagination:locale": "en_US"
    }
  ]
}
```

REQUEST

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   member-id
    Direction: -
    Offset:    -
    Limit:     -
    Locale:    invalid
```

RESPONSE

```
   error-type: application
   error-tag: invalid-value
   error-app-tag: ietf-list-pagination:locale-unavailable

   REQUEST

   This example targets an "ordered-by user" list.

   Target: /example-social:members/member=alice/favorites/uint8-numbers
     Pagination Parameters:
       Where:     -
       Sort-by:   .
       Direction: -
       Offset:    -
       Limit:     -
       Locale:    sv_SE

   RESPONSE

   error-type: application
   error-tag: invalid-value

   REQUEST

   This example supplies "locale" but not "sort-by".

   Target: /example-social:members/member
     Pagination Parameters:
       Where:     -
       Sort-by:   -
       Direction: -
       Offset:    -
       Limit:     -
       Locale:    sv_SE

   RESPONSE

   error-type: application
   error-tag: invalid-value
```

A.3.8. The "sublist-limit" Parameter

The "sublist-limit" parameter may be used on any target node.

A.3.8.1. target is a list entry

This example uses the target node '/example-social:members/member=alice' in the <intended> datastore.

> The target node is a specific list entry/element node, not the YANG "list" node.

This example sets the sublist-limit value '1', which returns just the first entry for all descendant lists and leaf-lists.

Note that, in the response, the "remaining" metadata value is set on the first element of each descendant list and leaf-list having more than one value.

REQUEST

```
Datastore: <intended>
Target: /example-social:members/member=alice
Sublist-limit: 1
Pagination Parameters:
  Where:     -
  Sort-by:   -
  Direction: -
  Offset:    -
  Limit:     -
```

RESPONSE

```
{
  "example-social:member": [
    {
      "member-id": "alice",
      "email-address": "alice@example.com",
      "password": "$0$1543",
      "avatar": "BASE64VALUE=",
      "tagline": "Every day is a new day",
      "privacy-settings": {
        "hide-network": "false",
        "post-visibility": "public"
      },
      "following": ["bob"],
      "@following": [
        {
          "ietf-list-pagination:remaining": "2"
        }
      ],
      "posts": {
        "post": [
          {
            "@": {
              "ietf-list-pagination:remaining": "1"
            },
            "timestamp": "2020-07-08T13:12:45Z",
            "title": "My first post",
            "body": "Hiya all!"
          }
        ]
      },
      "favorites": {
        "uint8-numbers": [17],
        "int8-numbers": [-5],
        "@uint8-numbers": [
          {
            "ietf-list-pagination:remaining": "5"
          }
        ],
        "@int8-numbers": [
          {
            "ietf-list-pagination:remaining": "5"
          }
        ]
      }
    }
  ]
}
```

A.3.8.2. target is a datastore

This example uses the target node <intended> datastore.

This example sets the sublist-limit value '1', which returns just the first entry for all descendant lists and leaf-lists.

Note that, in the response, the "remaining" metadata value is set on the first element of each descendant list and leaf-list having more than one value.

REQUEST

```
Datastore: <intended>
Target: /
Sublist-limit: 1
Pagination Parameters:
  Where:     -
  Sort-by:   -
  Direction: -
  Offset:    -
  Limit:     -
```

RESPONSE

```
{
  "example-social:members": {
    "member": [
      {
        "@": {
          "ietf-list-pagination:remaining": "4"
        },
        "member-id": "bob",
        "email-address": "bob@example.com",
        "password": "$0$1543",
        "avatar": "BASE64VALUE=",
        "tagline": "Here and now, like never before.",
        "posts": {
          "post": [
            {
              "@": {
                "ietf-list-pagination:remaining": "2"
              },
              "timestamp": "2020-08-14T03:32:25Z",
              "body": "Just got in."
            }
          ]
        },
        "favorites": {
          "decimal64-numbers": ["3.14159"],
          "@decimal64-numbers": [
            {
              "ietf-list-pagination:remaining": "1"
            }
          ]
        }
      }
    ]
  }
}
```

A.3.9. Combinations of Parameters

A.3.9.1. All six parameters at once

REQUEST

```
     Datastore: <operational>
     Target: /example-social:members/member
     Sublist-limit: 1
     Pagination Parameters:
       Where:     stats/joined[starts-with(timestamp,'2020')]
       Sort-by:   member-id
       Direction: backwards
       Offset:    2
       Limit:     2

   RESPONSE

   {
     "example-social:member": [
       {
         "@": {
           "ietf-list-pagination:remaining": "1"
         },
         "member-id": "eric",
         "email-address": "eric@example.com",
         "password": "$0$1543",
         "avatar": "BASE64VALUE=",
         "tagline": "Go to bed with dreams; wake up with a purpose.",
         "following": ["alice"],
         "posts": {
           "post": [
             {
               "timestamp": "2020-09-17T18:02:04Z",
               "title": "Son, brother, husband, father",
               "body": "What's your story?"
             }
           ]
         },
         "favorites": {
           "bits": ["two"],
           "@bits": [
             {
               "ietf-list-pagination:remaining": "2"
             }
           ]
         },
         "stats": {
           "joined": "2020-09-17T19:38:32Z",
           "membership-level": "pro",
           "last-activity": "2020-09-17T18:02:04Z"
         }
       },
       {
```

```
        "member-id": "bob",
        "email-address": "bob@example.com",
        "password": "$0$1543",
        "avatar": "BASE64VALUE=",
        "tagline": "Here and now, like never before.",
        "posts": {
          "post": [
            {
              "@": {
                "ietf-list-pagination:remaining": "2"
              },
              "timestamp": "2020-08-14T03:32:25Z",
              "body": "Just got in."
            }
          ]
        },
        "favorites": {
          "decimal64-numbers": ["3.14159"],
          "@decimal64-numbers": [
            {
              "ietf-list-pagination:remaining": "1"
            }
          ]
        },
        "stats": {
          "joined": "2020-08-14T03:30:00Z",
          "membership-level": "standard",
          "last-activity": "2020-08-14T03:34:30Z"
        }
      }
    }
  }
```

Acknowledgements

This work has benefited from the discussions of RESTCONF resource collection over the years, in particular, [I-D.ietf-netconf-restconf-collection] which provides enhanced filtering features for the retrieval of data nodes with the GET method and [I-D.zheng-netconf-fragmentation] which document large size data handling challenge. The authors would like to thank the following for lively discussions on list (ordered by first name): Andy Bierman, Martin BjÃ¶rklund, Robert Varga, Rob Wills, and Rob Wilton. The authors would also like to thank Jen Linkova for the OPS Directorate review.

Authors' Addresses

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net

Qin Wu
Huawei Technologies
Email: bill.wu@huawei.com

Per Andersson
Ionio Systems
Email: per.ietf@ionio.se

Olof Hagsand
SUNET
Email: olof@hagsand.se

Hongwei Li
Hewlett Packard Enterprise
Email: flycoolman@gmail.com

NETCONF Working Group
Internet-Draft
Updates: 6241 (if approved)
Intended status: Standards Track
Expires: 3 October 2026

K. Watsen
Watsen Networks
Q. Wu
Huawei
P. Andersson
Ionio Systems
O. Hagsand
SUNET
H. Li
HPE

1 April 2026

NETCONF Extensions to Support List Pagination
draft-ietf-netconf-list-pagination-nc-11

Abstract

This document defines a mapping of the list pagination mechanism defined in [I-D.ietf-netconf-list-pagination] to NETCONF [RFC6241].

This document updates [RFC6241], to augment the <get> and <get-config> "rpc" statements, and [RFC8526], to augment the <get-data> "rpc" statement, to define input parameters necessary for list pagination.

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 3 October 2026.

Copyright Notice

Table of Contents

1. Introduction

This document defines a mapping of the list pagination mechanism defined in [I-D.ietf-netconf-list-pagination] to NETCONF [RFC6241].

This document updates [RFC6241] and [RFC8526], as described in Section 2.

While the pagination mechanism defined in this document is designed for the NETCONF protocol [RFC6241], the augmented RPCs MAY be used by the RESTCONF protocol [RFC8040] if the RESTCONF server implements the "ietf-list-pagination-nc" module.

The YANG data model in this document conforms to the Network Management Datastore Architecture defined in [RFC8342]

1.1. Terminology

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

1.2. Conventions

Various examples in this document use "BASE64VALUE=" as a placeholder value for binary data that has been base64 encoded (per Section 9.8 of [RFC7950]). This placeholder value is used because real base64 encoded structures are often many lines long and hence distracting to the example being presented.

2. Updates to NETCONF operations

2.1. Updates to RFC 6241

The <get> and <get-config> rpc statements are augmented to accept additional input parameters, as described in Section 3.

2.2. Updates to RFC 8526

The <get-data> rpc statement is augmented to accept additional input parameters, as described in in Section 3.

3. List Pagination for NETCONF

In order for NETCONF to support [I-D.ietf-netconf-list-pagination], this document extends the operations <get>, <get-config> and <get-data> to include additional input parameters and output annotations.

The updated operations accept a content filter parameter "where", similar to the "filter" parameter of <get-config>, but includes nodes for "list" and "leaf-list" filtering.

The "where" query parameter is used to specify the YANG list or leaf-list that is to be retrieved. This must be a path expression used to represent a list or leaf-list data node.

Prefixes in the "where" query parameter XPath expression MUST be the YANG module prefix. If the specified XPath expression is invalid, then an <rpc-error> is returned with error type value "application" and "error-tag" value "invalid-value".

The following tree diagram [RFC8340] illustrates the "ietf-netconf-list-pagination" module:

```
module: ietf-list-pagination-nc

  augment /nc:get/nc:input:
    +---w list-pagination
       +---w where?           yang:xpath1.0
       +---w locale?          string {sort}?
       +---w sort-by?         union {sort}?
       +---w direction?       enumeration
       +---w cursor?          string
       +---w offset?          uint32
       +---w limit?           union
       +---w sublist-limit?   union
  augment /nc:get-config/nc:input:
    +---w list-pagination
       +---w where?           yang:xpath1.0
       +---w locale?          string {sort}?
       +---w sort-by?         union {sort}?
       +---w direction?       enumeration
       +---w cursor?          string
       +---w offset?          uint32
       +---w limit?           union
       +---w sublist-limit?   union
  augment /ncds:get-data/ncds:input:
    +---w list-pagination
       +---w where?           yang:xpath1.0
       +---w locale?          string {sort}?
       +---w sort-by?         union {sort}?
       +---w direction?       enumeration
       +---w cursor?          string
       +---w offset?          uint32
       +---w limit?           union
       +---w sublist-limit?   union
```

Comments:

* This module augments three NETCONF "rpc" statements: get, get-config, and get-data.

* The "get" and "get-config" augments are against the YANG module defined in [RFC6241]. The "get-data" augment is against the YANG module defined in [RFC8526].

4. Error Reporting

When an input query parameter is supplied with an erroneous value, an <rpc-error> MUST be returned containing the error-type value "application", the error-tag value "invalid-value", and MAY include the error-severity value "error". Additionally the error-app-tag SHOULD be set containing query parameter specific error value.

4.1. The "offset" Query Parameter

If the "offset" query parameter value supplied is larger than the number of instances in the working result-set, the <rpc-error> MUST contain error-app-tag with value "offset-out-of-range".

5. YANG Module for List Pagination in NETCONF

The "ietf-netconf-list-pagination-nc" module defines conceptual definitions within groupings, which are not meant to be implemented as datastore contents by a server.

This module has normative references to [RFC6241], [RFC6243], [RFC6991], and [RFC8342].

<CODE BEGINS> file "ietf-list-pagination-nc@2026-04-02.yang"

```
module ietf-list-pagination-nc {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-list-pagination-nc";
  prefix lpgnc;

  import ietf-netconf {
    prefix nc;
    reference
      "RFC 6241: Network Configuration Protocol (NETCONF)";
  }

  import ietf-netconf-nmda {
    prefix ncds;
    reference
      "RFC 8526: NETCONF Extensions to Support the
                 Network Management Datastore Architecture";
  }

  import ietf-list-pagination {
```

```
  prefix lpg;
  reference
    "RFC XXXX: List Pagination for YANG-driven Protocols";
}

organization
  "IETF NETCONF (Network Configuration) Working Group";

contact
    "WG Web:   <https://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

     Author:   Kent Watsen
               <kent+ietf@watsen.net>

     Author:   Qin Wu
               <bill.wu@huawei.com>

     Author:   Per Andersson
               <per.ietf@ionio.se>

     Author:   Olof Hagsand
               <olof@hagsand.se>

     Author:   Hongwei Li
               <flycoolman@gmail.com>";

description
  "This module augments the <get>, <get-config>, and <get-data>
   'rpc' statements to support list pagination.

   Copyright (c) 2026 IETF Trust and the persons identified
   as authors of the code. All rights reserved.

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

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

   The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
   'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
   'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
```

```
        are to be interpreted as described in BCP 14 (RFC 2119)
        (RFC 8174) when, and only when, they appear in all
        capitals, as shown here.";

    revision 2026-04-02 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: NETCONF Extensions to Support List Pagination";
    }

    augment "/nc:get/nc:input" {
      description
        "Allow the 'get' operation to use content filter
         parameter for specifying the YANG list or leaf-list
         that is to be retrieved";
      uses lpg:pagination-parameters;
    }

    augment "/nc:get-config/nc:input" {
      description
        "Allow the 'get-config' operation to use content filter
         parameter for specifying the YANG list or leaf-list
         that is to be retrieved";
      uses lpg:pagination-parameters;
    }

    augment "/ncds:get-data/ncds:input" {
      description
        "Allow the 'get-data' operation to use content filter
         parameter for specifying the YANG list or leaf-list
         that is to be retrieved";
      uses lpg:pagination-parameters;
    }
  }

  <CODE ENDS>
```

6. IANA Considerations

6.1. The "IETF XML" Registry

This document registers one URI in the "ns" subregistry of the IETF XML Registry [RFC3688] maintained at https://www.iana.org/assignments/xml-registry/xml-registry.xhtml#ns. Following the format in [RFC3688], the following registration is requested:

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

6.2. The "YANG Module Names" Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020] maintained at https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml. Following the format defined in [RFC6020], the below registration is requested:

name: ietf-list-pagination-nc
namespace: urn:ietf:params:xml:ns:yang:ietf-list-pagination-nc
prefix: pgnc
RFC: XXXX

7. Security Considerations

7.1. The "ietf-netconf-list-pagination" YANG Module

This section follows the template defined in Section 3.7.1 of [RFC8407].

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

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

The security considerations for the base NETCONF protocol operations (see Section 9 of [RFC6241] and Section 6 of [RFC8526]) apply to the extension of operations <get>, <get-config>, and <get-data> defined in this document.

8. References

8.1. Normative References

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

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

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

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

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <https://www.rfc-editor.org/info/rfc6243>.

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

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

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

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

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

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

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

[I-D.ietf-netconf-list-pagination] Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "List Pagination for YANG-driven Protocols", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-10, 12 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-10>.

8.2. Informative References

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

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

[I-D.ietf-netconf-restconf-collection] Bierman, A., BjÃ¶rklund, M., and K. Watsen, "RESTCONF Collection Resource", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-collection-00, 30 January 2015, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-restconf-collection-00>.

[I-D.zheng-netconf-fragmentation] Zheng, G. and Z. Wang, "A NETCONF Extension for Data Fragmentation", Work in Progress, Internet-Draft, draft-zheng-netconf-fragmentation-02, 26 June 2018, <https://datatracker.ietf.org/doc/html/draft-zheng-netconf-fragmentation-02>.

Appendix A. Example YANG Module

The examples within this document use the "example-social" YANG module defined in Appendix A.1 of [I-D.ietf-netconf-list-pagination].

Appendix B. Example Data Set

The Example Data Set used by the examples is defined in Appendix A.2 of [I-D.ietf-netconf-list-pagination].

Appendix C. Example Queries

C.1. List pagination with all query parameters

This example mimics that Appendix A.3.9 of [I-D.ietf-netconf-list-pagination]. In this XML example, The xmlns:es attribute binds the "es" prefix to the example-social module name. The urn:ietf:params:xml:ns:yang:ietf-list-pagination-nc namespace is used for the list-pagination query parameters.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="42">
  <get>
    <source>
      <running/>
    </source>
    <filter type="xpath" select="/es:members/es:member"
      xmlns:es="https://example.com/ns/example-social"/>
    <list-pagination
      xmlns="urn:ietf:params:xml:ns:yang:ietf-list-pagination-nc">
      <where>//stats[starts-with(joined,'2020')]</where>
      <sort-by>joined</sort-by>
      <direction>backwards</direction>
      <offset>2</offset>
      <limit>2</limit>
      <sublist-limit>1</sublist-limit>
      <locale>sv_US</locale>
    </list-pagination>
  </get>
</rpc>
```

Response from the NETCONF server:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <members
```

```
        xmlns="https://example.com/ns/example-social"
        xmlns:lp="urn:ietf:params:xml:ns:yang:ietf-list-pagination">
        <member lp:remaining="1" lp:locale="en_US">
          <member-id>eric</member-id>
          <email-address>eric@example.com</email-address>
          <password>$0$1543</password>
          <avatar>BASE64VALUE=</avatar>
          <tagline>Go to bed with dreams; wake up with purpose.</tagli\
ne>
          <following>alice</following>
          <posts>
            <post>
              <timestamp>2020-09-17T18:02:04Z</timestamp>
              <title>Son, brother, husband, father</title>
              <body>What's your story?</body>
            </post>
          </posts>
          <favorites>
            <bits lp:remaining="2" lp:locale="en_US">two</bits>
          </favorites>
          <stats>
            <joined>2020-09-17T19:38:32Z</joined>
            <membership-level>pro</membership-level>
            <last-activity>2020-09-17T18:02:04Z</last-activity>
          </stats>
        </member>
        <member lp:remaining="1" lp:locale="en_US">
          <member-id>bob</member-id>
          <email-address>bob@example.com</email-address>
          <password>$0$1543</password>
          <avatar>BASE64VALUE=</avatar>
          <tagline>Here and now, like never before.</tagline>
          <posts>
            <post lp:remaining="2" lp:locale="en_US">
              <timestamp>2020-08-14T03:32:25Z</timestamp>
              <body>Just got in.</body>
            </post>
          </posts>
          <favorites>
            <decimal64-numbers lp:remaining="1" lp:locale="en_US">3.14\
159</decimal64-numbers>
          </favorites>
          <stats>
            <joined>2020-08-14T03:30:00Z</joined>
            <membership-level>standard</membership-level>
            <last-activity>2020-08-14T03:34:30Z</last-activity>
          </stats>
        </member>
```

```
      </members>
    </data>
  </rpc-reply>
```

Acknowledgements

This work has benefited from the discussions of RESTCONF resource collection over the years, in particular, [I-D.ietf-netconf-restconf-collection] which provides enhanced filtering features for the retrieval of data nodes with the GET method and [I-D.zheng-netconf-fragmentation] which document large size data handling challenge. The authors would like to thank the following for lively discussions on list (ordered by first name): Andy Bierman, Martin BjÃ¶rklund, Robert Varga, and Robert Wilton.

Authors' Addresses

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net

Qin Wu
Huawei
Email: bill.wu@huawei.com

Per Andersson
Ionio Systems
Email: per.ietf@ionio.se

Olof Hagsand
SUNET
Email: olof@hagsand.se

Hongwei Li
HPE
Email: flycoolman@gmail.com

NETCONF Working Group
Internet-Draft
Updates: 8040 (if approved)
Intended status: Standards Track
Expires: 16 August 2026

K. Watsen
Watsen Networks
Q. Wu
Huawei Technologies
P. Andersson
Ionio Systems
O. Hagsand
SUNET
H. Li
Hewlett Packard Enterprise

12 February 2026

RESTCONF Extensions to Support List Pagination
draft-ietf-netconf-list-pagination-rc-10

Abstract

This document defines a mapping of the list pagination mechanism defined in [I-D.ietf-netconf-list-pagination] to RESTCONF [RFC8040].

This document updates RFC 8040, to declare "list" and "leaf-list" as valid resource targets for the RESTCONF GET operation, to define GET query parameters necessary for list pagination, and to define a media-type for XML-based lists.

Table of Contents

1. Introduction

This document defines a mapping of the list pagination mechanism defined in [I-D.ietf-netconf-list-pagination] to RESTCONF [RFC8040].

This document updates RFC 8040, as described in Section 2.

Declaring "list" and "leaf-list" as valid resource targets for the GET operation is necessary for list pagination.

1.1. Terminology

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

1.2. Conventions

Various examples in this document use "BASE64VALUE=" as a placeholder value for binary data that has been base64 encoded (per Section 9.8 of [RFC7950]). This placeholder value is used because real base64 encoded structures are often many lines long and hence distracting to the example being presented.

2. Updates to RFC 8040

2.1. Resource Targets

This document extends Section 3.5 of [RFC8040] to add "list" and "leaf-list" nodes (not just their entries) as valid data resources for the "GET" operation.

2.2. Media Type

This document extends Section 3.2 of [RFC8040] to add a new media type, "application/yang-data+xml-list", to encode "list" and "leaf-list" nodes in XML.

The "application/yang-data+xml-list" media-type defines a pseudo top-level element called "xml-list" that is used to wrap the response set, thus ensuring that a single top-level element is returned for the XML encoding, as required by Section 4.3 of [RFC8040].

For JSON, the existing "application/yang-data+json" media type is sufficient, as the JSON format has built-in support for encoding arrays.

The "application/yang-data+xml-list" media type is registered in Section 3.2.1.

2.3. Query Parameters

This document extends Section 4.8 of [RFC8040] to add new query parameters "limit", "offset", "cursor", "direction", "sort-by", "locale", "where", and "sublist-limit".

These eight query parameters correspond to those defined in Sections 3.1 and 3.2 in [I-D.ietf-netconf-list-pagination].

Note that complex pagination with combination of query parameters may cause performance strain on constrained devices due to intensive processing on both the server and the client.

Name	Methods	Description
limit	GET, HEAD	Limits the number of entries returned. If not specified, the number of entries that may be returned is unbounded.
offset	GET, HEAD	Indicates the number of entries in the result set that should be skipped over when preparing the response. If not specified, then no entries in the result set are skipped.
cursor	GET, HEAD	Indicates where to start the working result set, the previous entries are skipped over. If not specified, then no entries in the result set are skipped over.
direction	GET, HEAD	Indicates the direction that the result set is to be traversed. If not specified, then the result set is traversed in the "forwards" direction.
sort-by	GET, HEAD	Indicates the node name that the result set should be sorted by. If not specified, then the result set's default order is used, per YANG's "ordered-by" statement.
locale	GET, HEAD	Specifies the locale the server should use when collating the result set. If not specified, the server chooses what locale is used for collation.
where	GET, HEAD	Specifies a filter expression that result set entries must match. If not specified, then no entries are filtered from the result set.
sublist-limit	GET, HEAD	Limits the number of entries returned returned for descendent lists and leaf-lists. If not specified, the number of entries that may be returned is unbounded.

For all of the query parameters, the query parameter is only allowed for the GET and HEAD methods on "list" and "leaf-list" data resources. A "400 Bad Request" status-line MUST be returned if used with any other method or resource type. The error-tag value "operation-not-supported" is used in this case.

Per the conformance defined in Section 3.1 of [I-D.ietf-netconf-list-pagination], all of these parameters MUST be supported for all "config true" lists and leaf-lists, and SHOULD be supported for "config false" lists and leaf-lists. A server MAY disable the support for some or all "config false" lists, as described in Section 3.3 of [I-D.ietf-netconf-list-pagination].

2.3.1. The "limit" Query Parameter

The "limit" query parameter corresponds to the "limit" parameter defined in Section 3.1.7 of [I-D.ietf-netconf-list-pagination].

If the limit value is invalid, i.e. not an unsigned 32-bit integer greater than or equal to 1 or the string "unbounded", then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.2. The "offset" Query Parameter

The "offset" query parameter corresponds to the "offset" parameter defined in Section 3.1.5 of [I-D.ietf-netconf-list-pagination].

If the offset value is invalid, a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

If the offset value exceeds the number of entries in the working result set, then a "416 Range Not Satisfiable" status-line MUST be returned with the error-type value "application", error-tag value "invalid-value", and SHOULD also include the "offset-out-of-range" identity as the error-app-tag value.

2.3.3. The "cursor" Query Parameter

The "cursor" query parameter corresponds to the "cursor" parameter defined in Section 3.1.6 of [I-D.ietf-netconf-list-pagination]. As described in Section 3.1.5 of [I-D.ietf-netconf-list-pagination] the server does not keep any stateful information about the "next" and "previous" cursor or the current page. Due to their ephemeral nature, cursor values are never cached.

If the cursor value is unknown, i.e. the key does not exist, a "404 Not Found" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value", and SHOULD also include the "cursor-not-found" identity as the error-app-tag value.

If the "cursor" query parameter is not supported on the target node, then a "501 Not Implemented" status-line MUST be returned with error-type value "application" and error-tag value "operation-not-supported".

2.3.4. The "direction" Query Parameter

The "direction" query parameter corresponds to the "direction" parameter defined in Section 3.1.4 of [I-D.ietf-netconf-list-pagination].

If the direction value is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.5. The "sort-by" Query Parameter

The "sort-by" query parameter corresponds to the "sort-by" parameter defined in Section 3.1.2 of [I-D.ietf-netconf-list-pagination].

If the specified node identifier is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.6. The "locale" Query Parameter

The "locale" query parameter corresponds to the "locale" parameter defined in Section 3.1.3 of [I-D.ietf-netconf-list-pagination].

If the specified node identifier is invalid, i.e. the locale is unknown to the server, then a "501 Not Implemented" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value", and SHOULD also include the "locale-unavailable" identity as the error-app-tag value.

If "locale" is supplied but not "sort-by", a "400 Bad Request" status-line MUST be returned with the error-type "application" and error-tag value "invalid-value".

2.3.7. The "where" Query Parameter

The "where" query parameter corresponds to the "where" parameter defined in Section 3.1.1 of [I-D.ietf-netconf-list-pagination].

Prefixes in the XPath expression MUST be YANG module names.

If the specified XPath expression is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.8. The "sublist-limit" Query Parameter

The "sublist-limit" query parameter corresponds to the "sublist-limit" parameter defined in Section 3.2.1 of [I-D.ietf-netconf-list-pagination].

If the sublist-limit value is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

3. IANA Considerations

3.1. The "RESTCONF Capability URNs" Registry

This document registers six capabilities in the RESTCONF Capability URNs [RFC8040] maintained at https://www.iana.org/assignments/restconf-capability-urns/restconf-capability-urns.xhtml. Following the instructions defined in Section 11.4 of [RFC8040], the below registrations are requested:

All registrations are to use this document (RFC XXXX) for the "Reference" value.

Index	Capability Identifier
:limit	urn:ietf:params:restconf:capability:limit:1.0
:offset	urn:ietf:params:restconf:capability:offset:1.0
:cursor	urn:ietf:params:restconf:capability:cursor:1.0
:direction	urn:ietf:params:restconf:capability:direction:1.0
:sort-by	urn:ietf:params:restconf:capability:sort-by:1.0
:locale	urn:ietf:params:restconf:capability:locale:1.0
:where	urn:ietf:params:restconf:capability:where:1.0
:sublist-limit	urn:ietf:params:restconf:capability:sublist-limit:1.0

3.2. The "Media Types" Registry

This document registers one media type in the "application" subregistry of the Media Types registry [RFC6838] [RFC4855] maintained at https://www.iana.org/assignments/media-types/media-types.xhtml#application. Following the format defined in [RFC4855], the below registration is requested:

3.2.1. Media Type "application/yang-data+xml-list"

Type name: application

Subtype name: yang-data+xml-list

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit
Each conceptual YANG data node is encoded according to the XML Encoding Rules and Canonical Format for the specific YANG data node type defined in [RFC7950].

Security considerations: Security considerations related to the generation and consumption of RESTCONF messages are discussed in Section 12 of RFC 8040. Additional security considerations are specific to the semantics of particular YANG data models. Each YANG module is expected to specify security considerations for the YANG data defined in that module.

Interoperability considerations: RFC XXXX specifies the format of conforming messages and the interpretation thereof.

Published specification: RFC XXXX

Applications that use this media type: Instance document data parsers used within a protocol or automation tool that utilize the YANG Patch data structure.

Fragment identifier considerations: Fragment identifiers for this type are not defined. All YANG data nodes are accessible as resources using the path in the request URI.

Additional information:

Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): None
Macintosh file type code(s): "TEXT"

Person & email address to contact for further information:
See the Authors' Addresses section of RFC XXXX.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC XXXX.

Change controller: Internet Engineering Task Force
(mailto:iesg@ietf.org).

Provisional registration? (standards tree only): no

4. Security Considerations

This document introduces protocol operations for paging through data already provided by the RESTCONF protocol, and hence does not introduce any new security considerations.

This document does not define a YANG module and hence there are no data modeling considerations beyond those discussed in [I-D.ietf-netconf-list-pagination].

5. References

5.1. Normative References

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

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

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

[I-D.ietf-netconf-list-pagination] Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "List Pagination for YANG-driven Protocols", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-09, 12 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-09>.

5.2. Informative References

[RFC4855] Casner, S., "Media Type Registration of RTP Payload Formats", RFC 4855, DOI 10.17487/RFC4855, February 2007, <https://www.rfc-editor.org/info/rfc4855>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

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

[I-D.ietf-netconf-restconf-collection] Bierman, A., BjÃ¶rklund, M., and K. Watsen, "RESTCONF Collection Resource", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-collection-00, 30 January 2015, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-restconf-collection-00>.

[I-D.zheng-netconf-fragmentation] Zheng, G. and Z. Wang, "A NETCONF Extension for Data Fragmentation", Work in Progress, Internet-Draft, draft-zheng-netconf-fragmentation-02, 26 June 2018, <https://datatracker.ietf.org/doc/html/draft-zheng-netconf-fragmentation-02>.

Appendix A. Example YANG Module

The examples within this document use the "example-social" YANG module defined in Appendix A.1 of [I-D.ietf-netconf-list-pagination].

Appendix B. Example Data Set

The Example Data Set used by the examples is defined in Appendix A.2 of [I-D.ietf-netconf-list-pagination].

Appendix C. Example Queries

C.1. List pagination with all query parameters

This example mimics that Appendix A.3.9 of [I-D.ietf-netconf-list-pagination]. This example is presented twice, once using XML and again using JSON.

XML:

```
   =============== NOTE: '\' line wrapping per RFC 8792 ================

   GET /restconf/ds/ietf-datastores:operational/example-social:members/\
   member?where=//stats//joined[starts-with(timestamp,'2020')]&sort-by=\
   timestamp&direction=backwards&offset=2&limit=2&sublist-limit=1 HTTP/\
   1.1
   Host: example.com
   Accept: application/yang-data+xml-list

   Response from the RESTCONF server:

   =============== NOTE: '\' line wrapping per RFC 8792 ================

   HTTP/1.1 200 OK
   Date: Thu, 26 Jan 2017 20:56:30 GMT
   Server: example-server
   Last-Modified: Thu, 26 Jan 2017 20:55:30 GMT
   Content-Type: application/yang-data+xml-list

   <xml-list>
     <member
       xmlns="https://example.com/ns/example-social"
       xmlns:lp="urn:ietf:params:xml:ns:yang:ietf-list-pagination"
       lp:remaining="1" lp:locale="en_US">
       <member-id>eric</member-id>
       <email-address>eric@example.com</email-address>
       <password>$0$1543</password>
       <avatar>BASE64VALUE=</avatar>
       <tagline>Go to bed with dreams; wake up with purpose.</tagline>
       <following>alice</following>
       <posts>
         <post>
           <timestamp>2020-09-17T18:02:04Z</timestamp>
           <title>Son, brother, husband, father</title>
           <body>What's your story?</body>
         </post>
       </posts>
       <favorites>
         <bits lp:remaining="2" lp:locale="en_US">two</bits>
       </favorites>
       <stats>
         <joined>2020-09-17T19:38:32Z</joined>
         <membership-level>pro</membership-level>
         <last-activity>2020-09-17T18:02:04Z</last-activity>
       </stats>
     </member>
     <member
       xmlns="https://example.com/ns/example-social"
```

```
      xmlns:lp="urn:ietf:params:xml:ns:yang:ietf-list-pagination"
      lp:remaining="1" lp:locale="en_US">
      <member-id>bob</member-id>
      <email-address>bob@example.com</email-address>
      <password>$0$1543</password>
      <avatar>BASE64VALUE=</avatar>
      <tagline>Here and now, like never before.</tagline>
      <posts>
        <post lp:remaining="2" lp:locale="en_US">
          <timestamp>2020-08-14T03:32:25Z</timestamp>
          <body>Just got in.</body>
        </post>
      </posts>
      <favorites>
        <decimal64-numbers lp:remaining="1" lp:locale="en_US">3.14159<\
/decimal64-numbers>
      </favorites>
      <stats>
        <joined>2020-08-14T03:30:00Z</joined>
        <membership-level>standard</membership-level>
        <last-activity>2020-08-14T03:34:30Z</last-activity>
      </stats>
    </member>
  </xml-list>

  JSON:

  =============== NOTE: '\' line wrapping per RFC 8792 ================

  GET /restconf/ds/ietf-datastores:running/example-social:members/memb\
  er?where=//stats//joined[starts-with(timestamp,'2020')]&sort-by=time\
  stamp&direction=backwards&offset=2&limit=2&sublist-limit=1 HTTP/1.1
  Host: example.com
  Accept: application/yang-data+json

  Response from the RESTCONF server:

  HTTP/1.1 200 OK
  Date: Thu, 26 Jan 2017 20:56:30 GMT
  Server: example-server
  Last-Modified: Thu, 26 Jan 2017 20:55:30 GMT
  Content-Type: application/yang-data+json

  {
    "example-social:member": [
      {
        "@": {
          "ietf-list-pagination:remaining": 1,
```

```
          "ietf-list-pagination:locale": "en_US"
        },
        "member-id": "eric",
        "email-address": "eric@example.com",
        "password": "$0$1543",
        "avatar": "BASE64VALUE=",
        "tagline": "Go to bed with dreams; wake up with purpose.",
        "following": ["alice"],
        "posts": {
          "post": [
            {
              "@": {
                "ietf-list-pagination:remaining": 2,
                "ietf-list-pagination:locale": "en_US"
              },
              "timestamp": "2020-09-17T18:02:04Z",
              "title": "Son, brother, husband, father",
              "body": "What's your story?"
            }
          ]
        },
        "favorites": {
          "bits": ["two"],
          "@example-social:bits": [
            {
              "ietf-list-pagination:remaining": 2,
              "ietf-list-pagination:locale": "en_US"
            }
          ]
        },
        "stats": {
          "joined": "2020-09-17T19:38:32Z",
          "membership-level": "pro",
          "last-activity": "2020-09-17T18:02:04Z"
        }
      },
      {
        "member-id": "bob",
        "email-address": "bob@example.com",
        "password": "$0$1543",
        "avatar": "BASE64VALUE=",
        "tagline": "Here and now, like never before.",
        "posts": {
          "post": [
            {
              "@": {
                "ietf-list-pagination:remaining": 2,
                "ietf-list-pagination:locale": "en_US"
```

```
              },
              "timestamp": "2020-08-14T03:32:25Z",
              "body": "Just got in."
            }
          ]
        },
        "favorites": {
          "decimal64-numbers": ["3.14159"],
          "@example-social:decimal64-numbers": [
            {
              "ietf-list-pagination:remaining": 1,
              "ietf-list-pagination:locale": "en_US"
            }
          ]
        },
        "stats": {
          "joined": "2020-08-14T03:30:00Z",
          "membership-level": "standard",
          "last-activity": "2020-08-14T03:34:30Z"
        }
      }
    ]
  }
```

Acknowledgements

This work has benefited from the discussions of RESTCONF resource collection over the years, in particular, [I-D.ietf-netconf-restconf-collection] which provides enhanced filtering features for the retrieval of data nodes with the GET method and [I-D.zheng-netconf-fragmentation] which document large size data handling challenge. The authors would like to thank the following for lively discussions on list (ordered by first name): Andy Bierman, Martin BjÃ¶rklund, Robert Varga, and Rob Wilton.

Authors' Addresses

Kent Watsen
Watsen Networks
Email: kent+ietf@watsen.net

Qin Wu
Huawei Technologies
Email: bill.wu@huawei.com

Per Andersson
Ionio Systems
Email: per.ietf@ionio.se

Olof Hagsand
SUNET
Email: olof@hagsand.se

Hongwei Li
Hewlett Packard Enterprise
Email: flycoolman@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 29 December 2026

J. Dai
Fiberhome/CICT
S. Yu
PCL
W. Cheng
China Mobile
M. Blanchet
Viagenie
P. Andersson
Ionio Systems
27 June 2026

NETCONF over QUIC
draft-ietf-netconf-over-quic-09

Abstract

This document specifies how to use QUIC as a secure transport for exchanging Network Configuration Protocol (NETCONF) messages. NETCONF over QUIC allows to take advantage of QUIC streams, for example, to eliminate some TCP head-of-line blocking issues. NETCONF over QUIC provides security properties similar to NETCONF over TLS.

This document also defines a YANG module which augments the ietf-netconf-client and ietf-netconf-server YANG modules.

Editorial note (to be removed by the RFC Editor

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

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

* AAAA --> the assigned RFC value for this draft
* BBBB --> the assigned RFC value for draft-ietf-netconf-netconf-client-server
* CCCC --> the assigned RFC value for draft-ietf-netconf-quic-client-server

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Network Configuration Protocol (NETCONF) [RFC6241] defines a mechanism through which the configuration of network devices can be installed, manipulated, and deleted.

NETCONF can be conceptually partitioned into four layers: content, operation, message and security transport layers.

The Secure Transport layer provides a communication path between the client and server. NETCONF can be layered over any transport protocol that provides a set of basic requirements, such as:

1. NETCONF is connection-oriented, requiring a persistent connection between peers. This connection MUST provide reliable and sequenced data delivery. NETCONF connections are long-lived, persisting between protocol operations.

2. NETCONF connections MUST provide authentication, data integrity, confidentiality, and replay protection. NETCONF depends on the transport protocol for this capability.

The NETCONF protocol is not bound to any particular transport protocol, but allows a mapping to define how it can be implemented over any specific protocol.

However, because of the connection-oriented feature, almost all of the current secure transport protocols used by NETCONF are TCP based. As is well known, TCP has some shortcomings such as head-of-line blocking.

QUIC ([RFC9000][RFC9001]) conforms to the above requirements, therefore is also an appropriate transport protocol for NETCONF. Moreover, QUIC provides the following additional benefits not present in the other NETCONF transports:

* Single connection can be long lived and support multiple NETCONF RPC calls and responses within the same connection, using streams. This is very useful for a network management control station who is regularly monitoring devices and therefore having a long lived connection requires way less resources on both peers.

* 1 RTT initial handshake that includes TLS.

* Adaptable to more difficult environments such as those with long delays ([I-D.many-tiptop-usecase], [I-D.many-tiptop-quic-profile] .

Therefore, QUIC is a proper transport protocol for the secure transport layer of NETCONF. This document specifies how to use QUIC as the secure transport protocol for NETCONF.

2. Terminology and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Connection Management

3.1. Connection establishment

QUIC connections are established as described in [RFC9000]. During connection establishment, support is indicated by selecting the ALPN token registered for NETCONF over QUIC (see Section 12) in the cryptographic handshake.

3.1.1. Early data

The QUIC protocol uses TLS 1.3 messages to secure the transport. This means that Early data (aka 0-RTT data) is supported. [RFC9001]

Early data (aka 0-RTT data) is a mechanism defined in TLS 1.3 [I-D.ietf-tls-rfc8446bis] that allows a client to send data ("early data") as part of the first flight of messages to a server. Note that TLS 1.3 can be used without early data as per Appendix F.5 of [I-D.ietf-tls-rfc8446bis]. In fact, early data is permitted by TLS 1.3 only when the client and server share a Pre-Shared Key (PSK), either obtained externally or via a previous handshake. The client uses the PSK to authenticate the server and to encrypt the early data.

As noted in Section 2.3 of [I-D.ietf-tls-rfc8446bis], the security properties for early data are weaker than those for subsequent TLS-protected data. In particular, early data is not forward secret, and there is no protection against the replay of early data between connections. Appendix E.5 of [I-D.ietf-tls-rfc8446bis] requires applications not use early data without a profile that defines its use. This document specifies that NETCONF over QUIC implementations MUST NOT use early data.

3.2. Connection Termination

3.2.1. QUIC Connection Termination Process

The typical QUIC connection termination process is described in [RFC9000]

3.2.2. Considerations for Connection Termination

When a NETCONF session is implemented based on a QUIC connection, the idle timeout should be set appropriately in order to keep the QUIC connection persistent even if the NETCONF session is idle. In some cases, disabling it may be a possible option.

When a NETCONF server receives a <close-session> request, it will gracefully close the NETCONF session. The server SHOULD close the associated QUIC connection.

When a NETCONF entity receives a <kill-session> request for an open session, it SHOULD close the associated QUIC connection.

When a NETCONF entity is detecting the interruption of the QUIC connection, it SHOULD send a <close-session> request to the peer NETCONF entity.

When a stateless reset event occurs, nothing needs to be done by either the client or the server.

4. Stream mapping and usage

The NETCONF protocol layers specified in [RFC6241] are presented in Figure 1.

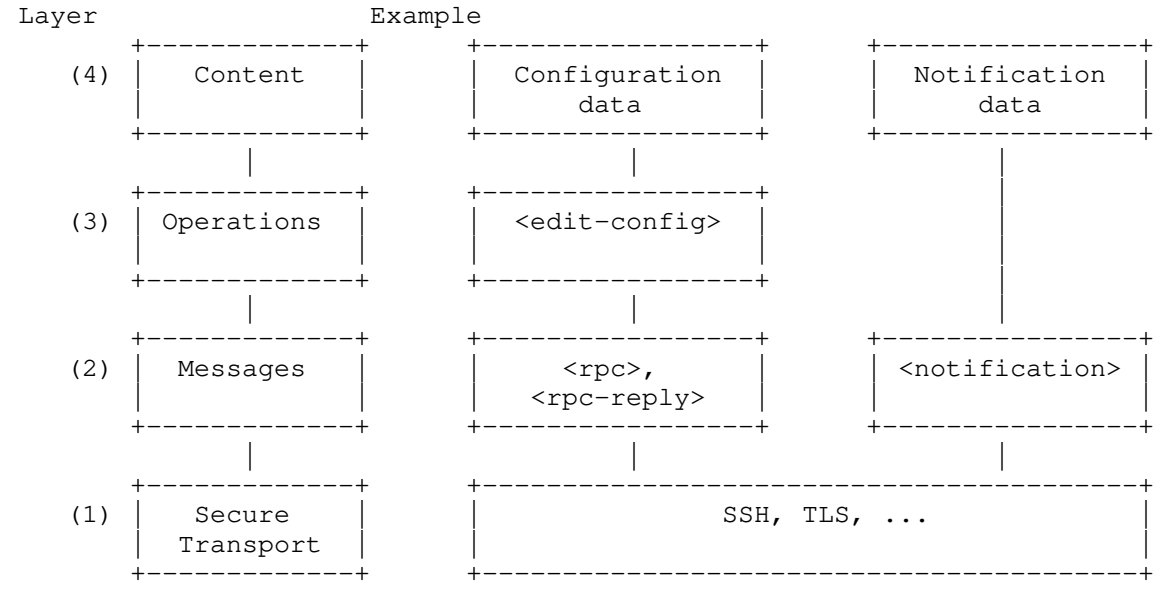


Figure 1: NETCONF Protocol Layers

Figure 1 shows that there are two kinds of main data flow exchanged between client and server:

* Configuration data from client to server.

* Notification data from server to client.

The two kinds of data flow need to be mapped into QUIC streams.

QUIC Streams provide a lightweight, ordered byte-stream abstraction to an application. Streams can be unidirectional or bidirectional meanwhile streams can be initiated by either the client or the server. Unidirectional streams carry data in one direction: from the initiator of the stream to its peer. Bidirectional streams allow for data to be sent in both directions.

QUIC uses Stream ID to identify the stream. The least significant bit (0x1) of the stream ID identifies the initiator of the stream. The second least significant bit (0x2) of the stream ID distinguishes between bidirectional streams (with the bit set to 0) and

unidirectional streams. There are four types of streams which are described in [RFC9000]. And Table 1 also describes the four types of streams

Acronym	Stream Type
C-BD	Client-Initiated, Bidirectional
S-BD	Server-Initiated, Bidirectional
C-UN	Client-Initiated, Unidirectional
S-UN	Server-Initiated, Unidirectional

Table 1: Stream Acronym Types

4.1. Bidirectional Stream Between client and server

NETCONF protocol uses an RPC-based communication model. Configuration data from client to server is exchanged based on '<rpc>' (the client initiating) and '<rpc-reply>' (sent by the server) and so on.

The messages used to exchange configuration data MUST be mapped into one bidirectional stream whose acronym is 'C-BD' according to Table 1. Since RPC processing is serialized and ordered within a session ([RFC6241] section 4.5), a bidirectional stream MUST be used for each NETCONF session.

4.2. Unidirectional Stream from server to client

There are some notification data exchanged between the client and the server. Notification is an server initiated message indicating that a certain event has been recognized by the server.

Notification messages are initiated by the server and no reply is needed from the client. So the messages used to exchange notification data MUST be mapped into one unidirectional stream whose acronym is 'S-UN' according to Table 1.

4.3. Mapping of QUIC connection, QUIC stream, and NETCONF session

The relationship among NETCONF sessions, QUIC streams and QUIC connections is illustrted as follows.

* One NETCONF session is allowed per QUIC connection.

* The NETCONF sessions, except subscriptions, runs over a QUIC bidi-stream.

* NETCONF Notifications and Subscribed Notifications runs over one QUIC uni-stream per subscription.

A subscription is initiated over the bidirectional stream by invoking either a "create-subscription" [RFC5277] or a "establish-subscription" [RFC8369] RPC. The server then creates a unidirectional stream towards the client and starts sending notifications on it.

Both the server and and client need to keep track of which subscription is linked to each stream, how this is implemented is out of scope for this document.

When a subscription is terminated the server notifies the client if applicable for the subscription then tears down the unidirectional stream.

5. Call home considerations

5.1. protocol-layering perspective

The Call Home mechanism is described in [RFC8071]. However, [RFC8071] is based on SSH and TLS, and they use TCP as transport protocol. Since QUIC is the transport protocol for NETCONF over QUIC, implementation of Call Home for NoQ is also different. Since QUIC embeds TLS 1.3 records in the packets, a separate TLS record layer can be omitted (see [RFC9001]).

The following diagram illustrates call home from a protocol-layering perspective based on QUIC:

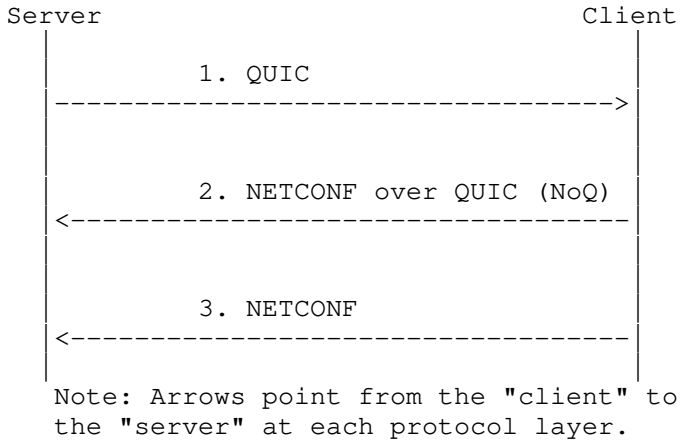


Figure 2: Call Home Sequence Diagram based on QUIC

This diagram makes the following points:

* The NETCONF server begins by initiating a QUIC connection to the NETCONF client.

* Using this QUIC connection, the NETCONF client initiates a NETCONF session to the NETCONF server.

Detailed description of the 'call home' mechanism is out of the scope of this document.

5.2. RFC8071 Call Home Specific Case

In the case of [RFC8071] Call home feature, where the NETCONF server initiates the transport connection to the NETCONF client, Table 1 will be used as follows: - the Client, referred in the Table, means the QUIC initiating party, therefore the NETCONF server and - the Server means the QUIC receiving party, therefore the NETCONF client.

6. Endpoint Authentication

Since QUIC uses TLS 1.3 this is used to verify server identity and client identity.

6.1. Server Identity

A server's identity MUST be verified according to Section 6 of [RFC7589].

6.2. Client Identity

A client's identity MUST be verified according to Section 7 of [RFC7589].

7. Framing

In order to mitigate delimiter injection attacks chunked framing as defined in [RFC6242] is required for NETCONF over QUIC.

The <hello> message MUST be followed by the character sequence RFC 5539 assumes that the end-of-message (EOM) sequence,]]>]]>>. Upon reception of the <hello> message, the receiving peer's QUIC layer conceptually passes the <hello> message to the Messages layer. If the :base:1.1 capability is advertised by both peers, the chunked framing mechanism defined in Section 4.2 of [[RFC6242]] is used for the remainder of the NETCONF session. Otherwise, the old end-of-message-based mechanism (see Section 4.3 of [[RFC6242]]) is used.

8. Overview of YANG Module

This document defines one YANG module that augments the NETCONF YANG groupings [I-D.ietf-netconf-netconf-client-server] with the QUIC transport YANG groupings [I-D.ietf-netconf-quic-client-server]. This section presents an overview of the YANG Module.

8.1. The "netconf-client" augmentation

The following tree diagram [RFC8340] illustrates the augmentation of the QUIC client grouping into the NETCONF client container:

```
  augment /ncc:netconf-client/ncc:initiate/ncc:netconf-server
            /ncc:endpoints/ncc:endpoint/ncc:transport:
    +--:(quic) {quic-initiate}?
       +--rw quic
          +---u quicc:quic-client
```

Figure 3

Comments:

* This augmentation to the "ncc:transport" container in "ietf-netconf-client.yang" adds a "quic" case with a "quic" container which uses the "quicc:quic-client" grouping.

* Note that the if-feature "quic-initiate" conditions if the "quic" container is available in the schema.

8.2. The "netconf-server" augmentation

The following tree diagram [RFC8340] illustrates the augmentation of the QUIC server grouping into the NETCONF server container:

```
  augment /ncs:netconf-server/ncs:listen/ncs:endpoints/ncs:endpoint
            /ncs:transport:
    +--:(quic) {quic-listen}?
       +--rw quic
          +---u quics:quic-server
```

Figure 4

Comments:

* This augmentation to the "ncs:transport" container in "ietf-netconf-server.yang" adds a "quic" case with a "quic" container which uses the "quics:quic-server" grouping.

* Note that the if-feature "quic-listen" conditions if the "quic" container is available in the schema.

9. YANG Module

This YANG module has normative references to [I-D.ietf-netconf-netconf-client-server], [I-D.ietf-netconf-quic-client-server], and [I-D.ietf-netmod-yang-semver].

<CODE BEGINS> file "ietf-netconf-quic@1.0.0-draft-ietf-netconf-over-quic-09.yang"

```
module ietf-netconf-quic {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-netconf-quic";
  prefix ncquic;

  import ietf-netconf-client {
    prefix ncc;
    reference
      "RFC BBBB: NETCONF Client and Server Models";
  }

  import ietf-netconf-server {
    prefix ncs;
    reference
      "RFC BBBB: NETCONF Client and Server Models";
```

```
  }

  import ietf-quic-client {
    prefix quicc;
    reference
      "RFC CCCC: YANG Groupings for QUIC Clients and QUIC Servers";
  }

  import ietf-quic-server {
    prefix quics;
    reference
      "RFC CCCC: YANG Groupings for QUIC Clients and QUIC Servers";
  }

  import ietf-yang-semver {
    prefix ysv;
    reference
      "draft-ietf-netmod-yang-semver: YANG Semantic Versioning";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:  https://datatracker.ietf.org/wg/netconf
     WG List: NETCONF WG list <mailto:netconf@ietf.org>

     Author:  Jinyou Dai
              <mailto:djy@fiberhome.se>

     Author:  Shaohua Yu
              <mailto:yush@cae.cn>

     Author:  Weiqiang Cheng
              <mailto:chengeiqiang@chinamobile.com>

     Author:  Marc Blanchet
              <mailto:marc.blanchet@viagenie.ca>

     Author:  Per Andersson
              <mailto:per.ietf@ionio.se>";

  description
    "This module defines augmentations for a NETCONF server to
     also support the QUIC transport.

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

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
        'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
        'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
        are to be interpreted as described in BCP 14 (RFC 2119)
        (RFC 8174) when, and only when, they appear in all
        capitals, as shown here.";

    revision 2026-06-27 {
      ysv:version 1.0.0-draft-ietf-netconf-over-quic-09;
      description
        "Initial version";
      reference
        "RFC AAAA: NETCONF over QUIC";
    }

    // Features

    feature quic-initiate {
      description
        "The 'quic-initiate' feature indicates that the NETCONF client
         supports initiating QUIC connections to NETCONF servers.";
      reference
        "RFC CCCC: YANG Groupings for QUIC Clients and QUIC Servers";
    }

    feature quic-listen {
      description
        "The 'quic-listen' feature indicates that the NETCONF server
         supports the QUIC transport.";
      reference
        "RFC AAAA: NETCONF over QUIC";
    }

    // Augments

    augment "/ncc:netconf-client/ncc:initiate" {
      if-feature "quic-initiate";
```

```
      description
        "Add 'quic-initate' feature to the NETCONF client connection
         configuration.";
    }

    augment "/ncc:netconf-client/ncc:initiate/ncc:netconf-server" +
            "/ncc:endpoints/ncc:endpoint/ncc:transport" {
      description
        "Add QUIC transport to the NETCONF client connection
         configuration";
      case quic {
        if-feature "quic-initiate";
        container quic {
          description
            "QUIC-level client parameters to initiate a NETCONF over
             QUIC connection.";
          uses quicc:quic-client;
        }
      }
    }

    augment "/ncs:netconf-server/ncs:listen" {
      if-feature "quic-listen";
      description
        "Add 'quic-listen' feature to the NETCONF server listen
         configuration.";
    }

    augment "/ncs:netconf-server/ncs:listen/ncs:endpoints" +
            "/ncs:endpoint/ncs:transport" {
      description
        "Add QUIC transport to the NETCONF server listen
         configuration.";
      case quic {
        if-feature "quic-listen";
        container quic {
          description
            "QUIC-level server parameters to listen for NETCONF over
             QUIC connections.";
          uses quics:quic-server;
        }
      }
    }
  }
```

Figure 5

<CODE ENDS>

10. Error codes

10.1. Transport Error Codes

Error codes of secure transport layer are specified in [[RFC9000]]. There are not new transport Error Codes defined for NETCONF over QUIC.

10.2. Application Error Codes

According to [[RFC9000]], management of application error codes is left to application protocols. and application error codes can be used by RESET_STREAM Frame and STOP_SENDING Frame. Application error codes for NETCONF over QUIC are listed as follows:

* NO_NETCONF PROTOCOL ERROR (0x00): no NETCONF errors happens.

* NETCONF CLOSE SESSION_ERROR (0x01): The peer tries to close a session which is not initiated by it.

* NETCONF CLOSE STREAM ERROR (0x02): The peer tries to close a bidirectional stream when the NETCONF session is active.

11. Security Considerations

The security considerations described throughout [RFC8446], [RFC6241], [RFC9000], [RFC9001], [RFC8071] and [I-D.ietf-netconf-netconf-client-server] apply here as well. "NoQ" requires verification of server identity and client identity according to [RFC7589].

"NoQ" relies on QUIC, which itself relies on TLS 1.3 and thus by defalut supports the protections against downgrade attacks described in [RFC9325]. QUIC specific issues and their mitigations are described in Section 21 of [RFC9000].

If invalid data or malformed messages are encountered, a robust implementation of this document MUST silently discard the message without further processing and then stop the NETCONF session.

12. IANA Considerations

This document creates a new registration for the identification of NETCONF over QUIC in the "Application Layer Protocol Negotiation (ALPN) Protocol IDs registry established in [RFC7301].

The "noq" string identifies NETCONF over QUIC:

* Protocol: NETCONF over QUIC

* Identification Sequence: 0x6e 0x6f 0x71 ("noq")

* Specification: This document

This document also requests IANA to reserve a UDP port for 'NETCONF over QUIC':

* Service Name: netconf-quic

* Transport Protocol(s): UDP

* Assignee: IESG iesg@ietf.org

* Contact: IETF Chair chair@ietf.org

* Description: NETCONF protocol over QUIC transport

* Reference: RFC AAAA

* Port number: 831

* Assignment Notes: Port 831 was assigned to netconf-beep, but a de-assignment is requested in [RFC9900].

* Service Name: netconf-ch-quic

* Transport Protocol(s): UDP

* Assignee: IESG iesg@ietf.org

* Contact: IETF Chair chair@ietf.org

* Description: NETCONF Call Home (QUIC)

* Reference: RFC AAAA

* Port number: 4335

13. Acknowledgements

The authors would like to acknowledge the contributors Yang Kou, Xueshun Wang, Kent Watsen, Jeffrey Haas, BalÃ¡zs Lengyel, Robert Wilton, Huaimo Chen, Lifen Zhou, Andy Bierman, Sean Turner, and Joe Clarke for their beneficial comments.

The authors would like to acknowledge the very useful feedback from an early implementor: Adolfo Ochagavia.

14. References

14.1. Normative References

[I-D.ietf-netconf-netconf-client-server]
Watsen, K., "A YANG Data Model for NETCONF Clients and Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-netconf-client-server-41, 4 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-netconf-client-server-41>.

[I-D.ietf-netconf-quic-client-server]
Andersson, P., "YANG Groupings for QUIC clients and QUIC servers", Work in Progress, Internet-Draft, draft-ietf-netconf-quic-client-server-07, 24 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-quic-client-server-07>.

[I-D.ietf-netmod-yang-semver]
Clarke, J., Wilton, R., Rahman, R., Lengyel, B., Sterne, J., and B. Claise, "YANG Semantic Versioning", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-semver-26, 7 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-26>.

[I-D.ietf-netmod-yang-module-filename]
Andersson, P., "YANG module file name convention", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-filename-14, 4 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-filename-14>.

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

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/info/rfc5277>.

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

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <https://www.rfc-editor.org/info/rfc8071>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8369] Kaplan, H., "Internationalizing IPv6 Using 128-Bit Unicode", RFC 8369, DOI 10.17487/RFC8369, April 2018, <https://www.rfc-editor.org/info/rfc8369>.

[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/info/rfc9000>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>.

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

14.2. Informative References

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC9900] Boucadair, M., "Updates to NETCONF Transport Port Numbers", RFC 9900, DOI 10.17487/RFC9900, December 2025, <https://www.rfc-editor.org/info/rfc9900>.

[I-D.many-tiptop-usecase] Blanchet, M., Eddy, W., and M. Eubanks, "IP in Deep Space: Key Characteristics, Use Cases and Requirements", Work in Progress, Internet-Draft, draft-many-tiptop-usecase-03, 18 June 2025, <https://datatracker.ietf.org/doc/html/draft-many-tiptop-usecase-03>.

[I-D.many-tiptop-quic-profile] Blanchet, M. and W. Eddy, "QUIC Profile for Deep Space", Work in Progress, Internet-Draft, draft-many-tiptop-quic-profile-02, 1 March 2026, <https://datatracker.ietf.org/doc/html/draft-many-tiptop-quic-profile-02>.

[I-D.ietf-tls-rfc8446bis] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-rfc8446bis-14, 13 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-14>.

Authors' Addresses

Jinyou Dai
Fiberhome Telecom LTD./CICT.
Gaoxin 4th Road 6#
Wuhan, Hubei 430079
China
Email: djy@fiberhome.com

Shaohua Yu
China PCL.
China
Email: yush@cae.cn

Weiqiang Cheng
China Mobile
China
Email: chengweiqiang@chinamobile.com

Marc Blanchet
Viagenie
Canada
Email: marc.blanchet@viagenie.ca

Per Andersson
Ionio Systems
Sweden
Email: per.ietf@ionio.se

Internet Engineering Task Force
Internet-Draft
Updates: 6241, 8526, 9144 (if approved)
Intended status: Standards Track
Expires: 24 August 2026

JG. Cumming
Nokia
R. Wills
Cisco Systems

20 February 2026

NETCONF and RESTCONF Private Candidate Datastores
draft-ietf-netconf-privcand-09

Abstract

This document provides a mechanism to extend the Network Configuration Protocol (NETCONF) and RESTCONF protocol to support multiple clients making configuration changes simultaneously and ensuring that they commit only those changes that they defined.

This document addresses two specific aspects: The interaction with a private candidate over the NETCONF and RESTCONF protocols and the methods to identify and resolve conflicts between clients.

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 24 August 2026.

Copyright Notice

Table of Contents

1. Introduction

NETCONF [RFC6241] and RESTCONF [RFC8040] both provide a mechanism for one or more clients to make configuration changes to a device running as a NETCONF/RESTCONF server. Each client has the ability to make one or more configuration changes to the server's shared candidate configuration.

As the name shared candidate suggests, all clients have access to the same candidate configuration. This means that multiple clients may make changes to the shared candidate prior to the configuration being committed. This behaviour may be undesirable as one client may unwittingly commit the configuration changes made by another client.

NETCONF provides a way to mitigate this behaviour by allowing clients to place a lock on the shared candidate. This lock prevents other clients from making changes until the lock is released, but this behaviour is, in many situations, also undesirable.

Many network devices support candidate configurations within the CLI interface, where a user (machine or otherwise) is able to edit a self-contained copy of a device's configuration without blocking other users from doing so.

This document specifies the extensions to the NETCONF protocol in order to support the use of private candidates. It also describes how the RESTCONF protocol can be used on a system that implements private candidates.

1.1. Requirements Language

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

1.2. Updates to RFC 6241 and RFC 8526

This document updates [RFC6241] to describe how the <edit-config>, <copy-config>, <get>, <get-config>, <commit>, <cancel-commit>, <delete-config>, <discard-changes>, <lock> and <unlock> operations work with a private candidate datastore. These updates are described in Section 3.8.2. This document also adds a new <update> operation, described in Section 3.8.1.

This document also updates [RFC8526] to describe how the <edit-data> and <get-data> operations work with a private candidate datastore. These updates are described in Section 3.8.2.

1.3. Updates to RFC 9144

This document updates [RFC9144] to augment the <compare> operation to describe how it works with the private candidate datastore. These updates are described in Section 3.8.2.7.

1.4. Limitations using the shared candidate configuration for multiple clients

The following sections describe some limitations and mitigation factors in more detail for the use of the shared candidate configuration during multi-client configuration over NETCONF or RESTCONF.

1.4.1. Issues

1.4.1.1. Unintended deployment of alternate users configuration changes

Consider the following scenario:

1. Client 1 modifies item A in the shared candidate configuration
2. Client 2 then modifies item B in the shared candidate configuration
3. Client 2 then issues a <commit> RPC

In this situation, both client 1 and client 2 configurations will be committed by client 2. In a machine-to-machine environment client 2 may not have been aware of the change to item A and, if they had been aware, may have decided not to proceed.

1.4.2. Mitigation strategies when using a shared candidate datastore

1.4.2.1. Locking the shared candidate configuration datastore

In order to resolve unintended deployment of alternate users configuration changes as described above NETCONF provides the ability to lock a datastore in order to restrict other users from editing and committed changes.

This does resolve the specific issue above, however, it introduces another issue. Whilst one of the clients holds a lock, no other client may edit the configuration. This will result in the client

failing and having to retry. Whilst this may be a desirable consequence when two clients are editing the same section of the configuration, where they are editing different sections this behaviour may hold up valid operational activity.

Additionally, a lock placed on the shared candidate configuration datastore must also lock the running configuration, otherwise changes committed directly into the running datastore may conflict.

Finally, this locking mechanism isn't available to RESTCONF clients.

1.4.2.2. Always use the running configuration datastore

The use of the running configuration datastore as the target for all configuration changes does not resolve any issues regarding blocking of system access in the case a lock is taken, nor does it provide a solution for multiple NETCONF and RESTCONF clients as each configuration change is applied immediately and the client has no knowledge of the current configuration at the point in time that they commenced the editing activity nor at the point they commit the activity.

1.4.2.3. Fine-grained (partial) locking

[RFC5717] describes a partial lock mechanism that can be used on specific portions of the shared candidate datastore.

Partial locking does not solve the issues of staging a set of configuration changes such that only those changes get committed in a commit operation, nor does it solve the issue of multiple clients editing the same parts of the configuration at the same time.

Partial locking additionally requires that the client is aware of any interdependencies within the servers YANG models in order to lock all parts of the tree.

Finally, partial locking is not widely implemented.

2. Definitions and terminology

2.1. Session-specific datastore

A session-specific datastore is a configuration datastore that is bound to the specific NETCONF session or RESTCONF request, unlike the shared candidate and running configuration datastores which have only one per system.

2.2. Shared candidate configuration

The candidate configuration datastore defined in [RFC6241] is referenced as the shared candidate configuration in this document.

2.3. Private candidate configuration

A private candidate configuration is a session-specific candidate configuration datastore.

When a private candidate is used by NETCONF, the specific session (and user) that created the private candidate configuration is the only session (user) that has access to it over NETCONF. Devices may expose this to other users through other interfaces but this is out of scope for this document.

When a private candidate is used by RESTCONF, the client that created the private candidate configuration is the only client that has access to it over RESTCONF.

The private candidate configuration contains a full copy of the running configuration when it is created (in the same way as a branch does in a source control management system and in the same way as the candidate configuration datastore as defined in [RFC6241]). When the private candidate datastore is in use, any operations that target the <candidate> configuration datastore (for example, <edit-config> and <edit-data> operations) actually make changes to the private candidate datastore instead of the shared candidate configuration datastore.

Obtaining this private candidate over NETCONF or RESTCONF will display the entire configuration, including all changes made to it. Performing a <commit> operation will merge the changes from the private candidate into the running configuration (the same as a merge in source code management systems). This merge is performed in two steps and is described further in Section 3.8.2.1. A <discard-changes> operation will revert the private candidate to the branch's initial state or it's state at the last <update> (whichever is most recent).

All changes made to this private candidate configuration are held separately from any other candidate configuration changes, whether made by other users to the shared candidate or any other private candidate, and are not visible to or accessible by any other configuration session.

3. Private candidates solution

To resolve the concurrency and locking issues described in Section 1.4 without sacrificing the benefits of a candidate datastore, this document introduces the concept of a private candidate: a session-specific datastore that isolates changes until they are ready to be committed.

NETCONF sessions and RESTCONF clients are able to utilize private candidates to streamline network operations, particularly for machine-to-machine communication.

Using this approach, clients may improve their performance and reduce the likelihood of blocking other clients from continuing with valid operational activities.

One or more private candidates may exist at any one time, however, a private candidate SHOULD:

* Be accessible by one client only

* Be visible by one client only

Additionally, the choice of using a shared candidate configuration datastore or a private candidate configuration datastore MUST be for the entire duration of the NETCONF session.

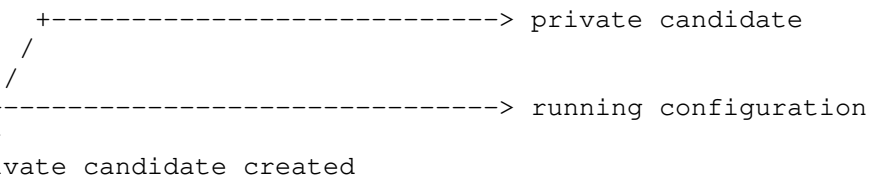
3.1. What is a private candidate

A private candidate is defined in the definitions and terminology (Section 2.3) section of this document.

3.2. When is a private candidate created

A private candidate datastore is created when the first RPC that requires access to it is sent to the server. This could be, for example, an <edit-config>.

When the private candidate is created a copy of the running configuration is made and stored in it. This can be considered the same as creating a branch in a source code repository.



3.3. When is a private candidate destroyed

A private candidate is valid for the duration of the NETCONF session, or the duration of the RESTCONF request. Issuing a <commit> operation will not close the private candidate but will issue an implicit <update> operation resyncing changes from the running configuration. More details on this later in this document.

A NETCONF session that is operating using a private candidate will discard all uncommitted changes in that session's private candidate and destroy the private candidate if the session is closed through a deliberate user action or disconnected for any other reason (such as a loss of network connectivity).

3.4. When is a private candidate updated

An <update> operation performed on a private candidate triggers a rebase of the private candidate configuration datastore against the running configuration datastore. This rebase is conceptually described as replacing the candidate configuration datastore with the current running configuration datastore at that point in time and replaying the configuration changes from the candidate configuration datastore against it, identifying any conflicts as this process progresses.

A server may choose to automatically issue an update when any client updates the running configuration datastore. Triggering an update in this manner must be specifically chosen and signalled by a server using an optional parameter to the :private-candidate NETCONF capability (see Section 3.5).

A <commit> operation always issues an implicit <update>, therefore it is not necessary for a client to perform an <update> before a <commit>. The implicit <update> will fail if conflicts are found.

More details on the <update> operation are provided later in this document.

3.5. How to signal the use of private candidates

3.5.1. Server

The server MUST signal its support for private candidates. The server does this by advertising a new :private-candidate capability:

urn:ietf:params:netconf:capability:private-candidate:1.0

As support for the shared candidate configuration datastore is a pre-requisite for the private candidate functionlity, a server MUST also advertise the :candidate capability as defined in [RFC6241].

If a server chooses to execute an update operation automatically when any client (not just the local client) performs an update to the running configuration datastore it must advertise this behaviour using the following optional parameter to the capability: trigger=all-updates.

3.5.2. NETCONF client

In order to utilise a private candidate configuration within a NETCONF session, the client must inform the server that it wishes to do this.

When a NETCONF client connects with a server it sends a list of client capabilities including one of the :base NETCONF version capabilties.

In order to enable private candidate mode for the duration of the NETCONF client session the NETCONF client sends the following capability:

urn:ietf:params:netconf:capability:private-candidate:1.0

In order for the use of private candidates to be established using this approach both the NETCONF server and the NETCONF client MUST advertise this capability.

If a client sends the capability and the server does not support private candidates, the server MUST choose one of the following options:

* Ignore the capability and continue with the session without the use of private candidates (this ensures backwards compatibility with older servers).

* Close the session as the requested functionality cannot be provided. New clients and servers implementing private candidate support should use this option.

When a server receives the client capability its mode of operation will be set to private candidate mode for the duration of the NETCONF session.

All RPC requests that target or impact the shared candidate configuration datastore will implicitly target or impact the session's private candidate configuration datastore instead, and operate in exactly the same way.

Using this method, the use of private candidates can be made available to NMDA and non-NMDA capable servers.

3.5.3. RESTCONF client

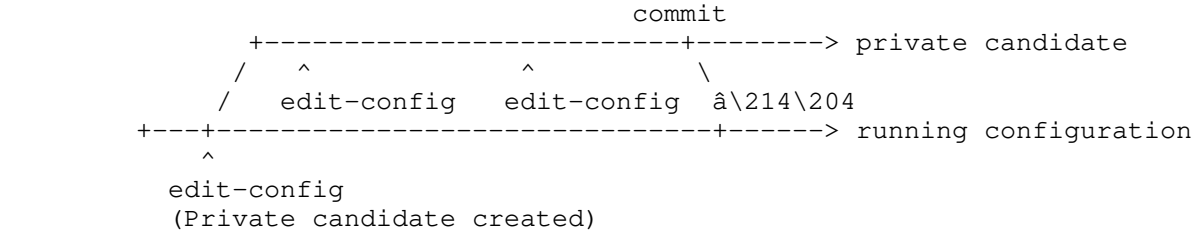
RESTCONF does not provide a mechanism for the client to advertise a capability. Therefore when a RESTCONF server advertises the :private-candidate capability, all client requests will use a private candidate when the client references the "{+restconf}/data" resource described in Section 3.3.1 of [RFC8040]. All edits are made to the client's private candidate, and the private candidate is automatically committed.

This ensures backwards compatibility with RESTCONF clients that are not aware of private candidates, because those clients will expect their changes to be committed immediately.

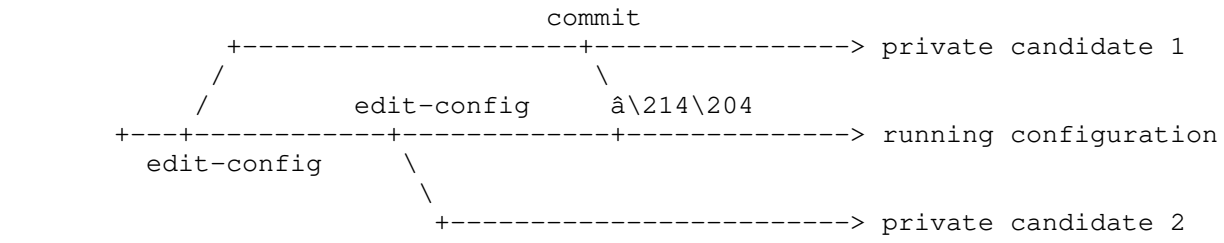
3.6. Interaction between running and private-candidate(s)

Multiple operations may be performed on the private candidate in order to stage changes ready for a commit.

In the simplest example, a session may create a private candidate configuration, perform multiple operations (such as <edit-config>) on it and then perform a <commit> operation to merge the private candidate configuration into the running configuration in line with semantics in [RFC6241].



More complex scenarios need to be considered, when multiple private candidate sessions are working on their own configuration (branches) and they make commits into the running configuration.



In this situation, if, how and when private candidate 2 is updated with the information that the running configuration has changed must be considered.

As described earlier, the client MUST be aware of changes to its private candidate configuration that are different from the running configuration datastore so it can be assured that it is only committing its own modifications.

It is possible, during an update, for conflicts to occur and the detection and resolution of these is discussed later in this document.

A good way to understand the interaction between candidates is to consider them as branches such as you might find in a source code management system.

Each private candidate is treated as a separate branch and changes made to the running configuration are not placed into a private candidate datastore except in one of the following situations:

* The client requests that the private candidate be refreshed using a new <update> operation

* <commit> is issued from the private candidate (which MUST automatically issue an <update> operation for that private candidate immediately prior to committing the configuration)

* An implementation chooses to perform an <update> operation after a change to the running configuration by any other client.

It is possible for a private candidate configuration to become significantly out of sync with the running configuration should the private candidate be open for a long time, however, most NETCONF configuration activities (between the first <edit-config>/<edit-data> and a <commit>) are short-lived.

To provide visibility into the differences between the private candidate configuration datastore a <compare> [RFC9144] operation MAY be performed against:

* The initial creation point of the private candidate's branch

* Against the last update point of the private candidate's branch

* Against the running configuration

An implementation may choose, optionally, to automatically perform an <update> operation on each private candidate configuration datastore after a change to the running configuration from another client. In this situation, the client will be unaware of these changes to its private candidate configuration datastore, and issuing a manual <update> operation will be a no-op.

3.7. Detecting and resolving conflicts

3.7.1. What is a conflict?

A conflict is when the intent of the client may have been different had it had a different starting point. In configuration terms, a conflict occurs when the same set of nodes in a configuration being altered by one user are changed between the start of the configuration preparation (the first <edit-config>/<edit-data> operation) and the conclusion of this configuration activity (terminated by a <commit> operation).

The situation where conflicts have the potential of occurring are when multiple configuration sessions are in progress and one session commits changes into the running configuration after the private candidate (branch) was created.

When this happens a conflict occurs for each node modified in the running configuration that is also modified in the private candidate configuration.

A node is considered modified if:

* There is a change of any value

* There is a change of existence (or otherwise) of any list entry

* There is a change to the order of any list items in a list configured as "ordered-by user"

* There is a change of existence (or otherwise) of a presence container

* There is a change of any component member of a leaf-list

* There is a change to the order of any items in a leaf-list configured as "ordered-by user"

* There is a change of existence (or otherwise) of a leaf

* There is a change to any YANG metadata associated with the node

A server MAY choose to add extra checks in addition to the list above.

If a server implements the transaction ID feature then this MAY be considered as part of detecting a conflict.

Examples of conflicts include:

* An interface has been deleted in the running configuration that existed when the private candidate was created. A change to a child node of this specific interface is made in the private candidate using the default merge operation would, instead of changing the child node, both recreate the interface and then set the child node.

* A leaf has been modified in the running configuration from the value that it had when the private candidate was created. The private candidate configuration changes that leaf to another value.

3.7.2. Detecting and reporting conflicts

A conflict can occur when an <update> operation is triggered. This can occur in a number of ways:

* Manually triggered by the <update> NETCONF operation

* Automatically triggered by the server running an <update> operation, such as when a <commit> operation is performed by the client in the private candidate session.

When a conflict is identified that node is internally marked by the server as "in conflict" in the private candidate. This "in conflict" status does not propagate back up the tree to the parent node(s). Each node in the ancestral tree is evaluated as in conflict or otherwise on its own merits. The "in conflict" marker remains until

the conflict is resolved on that node. The mechanism by which a server marks a node as "in conflict" is outside the scope of this draft.

When a conflict occurs:

* The client MUST be given the opportunity to re-evaluate its intent based on the new information. The resolution of the conflict may be manual or automatic depending on the server and client decision (discussed later in this document).

* A <commit> operation (that MUST trigger an automatic <update> operation immediately before) MUST fail irrespective of any system-wide default resolution-mode. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

* An <update> operation triggered by a client (excluding updates triggered by the <commit> operation) MUST fail unless the server has explicitly configured a system-wide resolution-mode of prefer-candidate or prefer-running (discussed later in this document).

The location of the conflict(s) should be reported as a list of xpaths and values.

Note: If a server implementation has chosen to automatically issue an <update> operation every time a change is made to the running configuration, the server will use the system-wide system resolution mode. If this resolution mode is prefer-candidate or prefer-running the conflicts will be resolved using those rules. If the resolution mode is set to revert-on-conflict the semantics are the same as the prefer-candidate method, however, all changes, whether in conflict or otherwise will be marked in the private candidate as "in-conflict". This means that any subsequent <commit> will fail until the client makes a conscious choice to resolve them.

3.7.3. Conflict resolution

Conflict resolution defines which configuration elements are retained when a conflict is resolved; those from the running configuration or those from the private candidate configuration.

When a conflict is detected in any client-triggered activity, the client MUST be informed. The client then has a number of options available to resolve the conflict:

* Edit the candidate configuration nodes marked as "in conflict". When a node marked as "in conflict" is subsequently edited the "in conflict" marker on that node is removed.

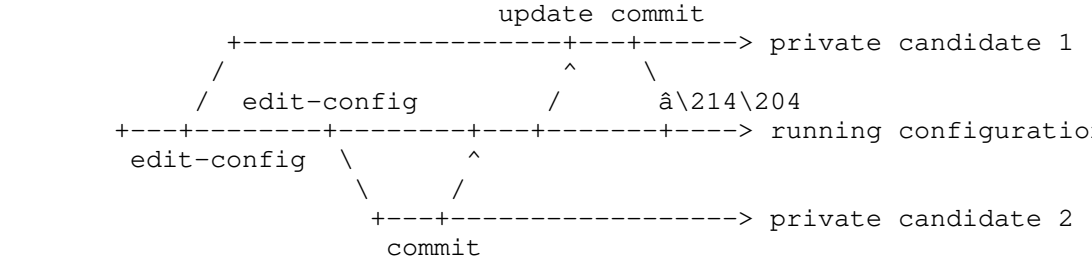
* Issue an <update> operation with a specific resolution-mode.

The <update> operation uses the resolution-mode specified in the request, or the system resolution mode in specific circumstances, to resolve the conflicts. The <update> operation is discussed later in this document.

The following configuration data is used below to illustrate the behaviour of each resolution method:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to London<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link to Tokyo<description>
    </interface>
  </interfaces>
</configure>
```

The example workflow is shown in this diagram and is used for the purpose of the examples below. In these examples the reader should assume that the <update> operation is manually provided by a client working in private candidate 1. The update operation triggered by a system upon commit is not shown.



There are three defined resolution methods:

3.7.3.1. Prefer-candidate

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 3.7.3.

When using the prefer-candidate resolution method, items in the running configuration that are not in conflict with the private candidate configuration are merged from the running configuration into the private candidate configuration. Nodes that are in conflict are ignored and not merged. The outcome of this is that the private candidate configuration reflects changes in the running that were not being worked on and those that are being worked on in the private candidate remain in the private candidate. Issuing a <commit> operation at this point will overwrite the running configuration with the conflicted items from the private candidate configuration.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and commits the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>prefer-candidate</resolution-mode>
  </update>
</rpc>
```

The un-conflicting changes are merged and the conflicting ones are ignored (and not merged from the running into private candidate 1).

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to San Francisco<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link moved to Paris<description>
    </interface>
  </interfaces>
</configure>
```

3.7.3.2. Prefer-running

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 3.7.3.

When using the prefer-running resolution method, items in the running configuration that are not in conflict with the private candidate configuration are merged from the running configuration into the private candidate configuration. Nodes that are in conflict are pushed from the running configuration into the private candidate configuration, overwriting any previous changes in the private candidate configuration. The outcome of this is that the private candidate configuration reflects the changes in the running configuration that were not being worked on as well as changing those being worked on in the private candidate to new values.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and commits the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>prefer-running</resolution-mode>
  </update>
</rpc>
```

The un-conflicting changes are merged and the conflicting ones are pushed into the private candidate 1 overwriting the existing changes.

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_two</name>
      <description>Link moved to Paris<description>
    </interface>
  </interfaces>
</configure>
```

3.7.3.3. Revert-on-conflict

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 3.7.3.

When using the revert-on-conflict resolution method, an update will fail to complete when any conflicting node is found. The session issuing the update will be informed of the failure.

No changes, whether conflicting or un-conflicting are merged into the private candidate configuration.

The owner of the private candidate session must then take deliberate and specific action to adjust the private candidate configuration to rectify the conflict. This may be by issuing further <edit-config> or <edit-data> operations, by issuing a <discard-changes> operation or by issuing an <update> operation with a different resolution method.

This resolution method MUST be the default resolution method as it provides for the highest level of visibility and control to ensure operational stability.

This resolution method MUST be supported by a server.

Example:

Session 1 edits the configuration by submitting the following:

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and commits the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>revert-on-conflict</resolution-mode>
  </update>
</rpc>
```

A conflict is detected, the update fails with an <rpc-error> and no merges/overwrite operations happen.

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to San Francisco<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link to Tokyo<description>
    </interface>
  </interfaces>
</configure>
```

3.7.4. System resolution mode and advertisement of this mode

The system resolution mode is revert-on-conflict. However, a system MAY choose to select a different system resolution mode.

The system resolution mode only takes effect if a server implementation performs an automatic update to all private candidate configurations following a commit to running.

The system resolution mode MUST be advertised in the :private-candidate capability by adding the default-resolution-mode parameter if the system default is anything other than revert-on-conflict. If the system default resolution mode is revert-on-conflict then advertising this in the :private-candidate capability is optional.

In this example, a server has configured a default system-wide resolution mode of prefer-running which MUST be signalled with the :private-candidate capability as follows:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?default-resolution-mode=prefer-running
```

3.7.5. Supported resolution modes

A server SHOULD support all three resolution modes. However, if the server does not support all three modes, the server MUST report the supported modes in the :private-candidate capability using the supported-resolution-modes, for example:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?supported-resolution-modes=revert-on-conflict,prefer-candidate
```

3.8. NETCONF operations

3.8.1. New NETCONF operations

3.8.1.1. <update>

The <update> operation triggers a rebase of the private candidate configuration against the running configuration. This rebase is conceptually described as replacing the candidate configuration with the current running configuration at that point in time and replaying the configuration changes from the candidate against it, identifing any conflicts as this process progresses.

The <update> operation may be triggered manually by the client or automatically by the server.

The <update> operation is atomic. This means that the entire <update> operation succeeeds or the entire <update> operation MUST fail.

The <update> operation MUST be implicitly triggered by a specific NETCONF session issuing a <commit> operation when using private candidates. This implicit <update> operation always has a resolution mode of revert-on-conflict. The actual order of operations in the server MUST be to issue the implicit <update> operation first and then the <commit> operation.

A <commit> operation that fails the implicit <update> operation MUST fail. The client is then required to make a specific decision to rectify the issue prior to committing. This may be to edit the private candidate configuration or to issue a manual <update> operation with a specific resolution mode selected.

3.8.1.1.1. <resolution-mode> parameter

The <update> operation takes the optional <resolution-mode> parameter

The resolution modes are described earlier in this document and the accepted inputs are:

* revert-on-conflict (default)
* prefer-candidate
* prefer-running

3.8.2. Updated NETCONF operations

Specific NETCONF operations altered by this document are listed in this section.

3.8.2.1. <commit>

The behaviour of the <commit> operation is updated such that processing a <commit> MUST follow a two step process in order to copy the proposed private configuration into the running configuration. These steps are:

1. An implicit update operation MUST be performed by the server using the resolution-mode revert-on-conflict (described earlier in this document).

2. On successful completion of step 1, the private candidate configuration is copied into the running configuration replacing the current contents.

3.8.2.1.1. Interactions with commit confirmed operations and private candidates

Nothing in this document alters the behaviour of the <confirmed>, <persist> or <persist-id> parameters and these MUST work when using the a private candidate configuration if the :confirmed-commit capability is advertised.

When a private candidate is committed using the <confirmed/> parameter and the commit operation disconnects the client's session, the configuration in the running configuration is immediately reverted and the proposed client changes are discarded.

When a private candidate is committed using the <confirmed/> parameter and the commit operation does not disconnect the client's session, and subsequently, the commit operation is either cancelled using the <cancel-commit> operation or the timeout expires, the running configuration is reverted and the proposed client changes are returned to the client's private candidate.

If a private candidate is committed using the <confirmed/> parameter and the <persist> parameter is provided, and the client subsequently disconnects its session for any reason whilst the timer is running, upon cancellation using the <cancel-commit> operation or on the expiry of the timer, the running configuration will be reverted, and the proposed client changes are discarded.

3.8.2.2. <get-config>

Performing a <get-config> operation with the candidate configuration datastore as the source input parameter will instead act on the private candidate datastore, and return data from that datastore. If no private candidate configuration has been created, one will be created at this point.

3.8.2.3. <edit-config>

Performing an <edit-config> operation with the candidate configuration datastore as the target, the changes will be placed into the private candidate configuration datastore. If no private candidate configuration has been created, one will be created at this point.

3.8.2.4. <copy-config>

When performing a <copy-config> operation with the candidate configuration datastore as the source or target, the private candidate will be used. If no private candidate configuration has been created, one will be created at this point.

3.8.2.5. <get-data>

Performing a <get-data> operation with the candidate configuration datastore as the datastore, the configuration from the private candidate will be returned. If no private candidate configuration has been created, one will be created at this point.

3.8.2.6. <edit-data>

Performing an <edit-data> operation with the candidate configuration datastore as the datastore, the changes will be placed into the private candidate configuration datastore. If no private candidate configuration has been created, one will be created at this point.

3.8.2.7. <compare>

Performing a <compare> [RFC9144] operation with the candidate datastore, operating as a private candidate, as either the <source> or <target> is a valid operation.

If <compare> is performed prior to a private candidate configuration being created, one will be created at that point.

The <compare> operation is extended by this document to allow the ability to compare the private candidate (at its current point in time) with the same private candidate at an earlier point in time, or with another datastore.

3.8.2.7.1. <reference-point> parameter

This document adds the optional <reference-point> node to the input of the <compare> operation that accepts the following values:

* last-update
* creation-point

Servers MAY support this functionality but are not required to by this document.

The last-update selection of <reference-point> will provide an output comparing the current private candidate configuration with the same private candidate configuration at the time it was last updated using the <update> NETCONF operation described in this document (whether automatically or manually triggered).

The creation-point selection of <reference-point> will provide an output comparing the current private candidate configuration datastore with the same private candidate configuration datastore at it was first created.

3.8.2.8. <lock>

The behaviour of the <lock> operation is updated such that locking the candidate configuration datastore will lock that session's private candidate configuration datastore only.

3.8.2.9. <unlock>

The behaviour of the <unlock> operation is updated such that unlocking the candidate configuration datastore will unlock that session's private candidate configuration datastore only.

3.8.2.10. <delete-config>

The behaviour of the <delete-config> operation is updated such that deleting the private candidate will destroy the private candidate for that session. A new one will subsequently be created on first access as described in Section 3.2.

3.8.2.11. <discard-changes>

The behaviour of the <discard-changes> operation is updated such that discarding the changes in a private candidate will reset it to the state it was when it was initially created, or to the state following the latest <update> operation, whichever is most recent. This state may not match the current running configuration.

To align the private candidate with the running configuration the <update> or <delete-config> operations may be used.

3.8.2.12. <get>

The <get> operation does not accept a datastore value and therefore this document is not applicable to this operation. The use of the get operation will not create a private candidate configuration.

3.8.2.13. <cancel-commit>

The <cancel-commit> operation is unchanged. Any changes made to the running configuration are returned to the private candidate if it still exists. See Section 3.8.2.1.1 for further details.

4. IANA Considerations

This document requests the registration the the following NETCONF capabilities:

* urn:ietf:params:netconf:capability:private-candidate:1.0 (Version 1.0)

This document also requests the registration of the following YANG modules in the YANG Module Names registry and in the IETF XML:

* ietf-netconf-private-candidate

* ietf-netconf-private-candidate-compare

5. Security Considerations

The "ietf-netconf-private-candidate" and "ietf-netconf-private-candidate-compare" YANG modules define data models that are designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These YANG-based management protocols (1) have to use a secure transport layer (e.g., SSH [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and (2) 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 preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

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:

* <update>: This operation modifies the private candidate configuration based on the current running configuration. Unauthorized access to this operation could lead to unintended configuration changes in the private candidate.

6. References

6.1. Normative References

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

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

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

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

[RFC5717] Lengyel, B. and M. Bjorklund, "Partial Lock Remote Procedure Call (RPC) for NETCONF", RFC 5717, DOI 10.17487/RFC5717, December 2009, <https://www.rfc-editor.org/info/rfc5717>.

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

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

6.2. Informative References

Appendix A. YANG modules

A.1. ietf-netconf-private-candidate@2026-02-03.yang

```
<CODE BEGINS> file "ietf-netconf-private-candidate@2026-02-03.yang"
module ietf-netconf-private-candidate {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-private-candidate";
  prefix pc;
```

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

     Editor:   James Cumming
               <james.cumming@nokia.com>

     Editor:   Robert Wills
               <rowills@cisco.com>";
  description
    "NETCONF private candidate support.

     Copyright (c) 2026 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

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

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

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
     NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
     'MAY', and 'OPTIONAL' in this document are to be interpreted as
     described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
     they appear in all capitals, as shown here.";

  revision 2026-02-03 {
    description
      "Introduce private candidate support";
    reference
      "draft-ietf-netconf-privcand:
       Netconf Private Candidates";
  }
  revision 2025-10-30 {
    description
      "Introduce private candidate support";
    reference
      "draft-ietf-netconf-privcand:
       Netconf Private Candidates";
  }
```

```
    revision 2024-09-12 {
      description
        "Introduce private candidate support";
      reference
        "draft-ietf-netconf-privcand:
         Netconf Private Candidates";
    }

    feature private-candidate {
      description
        "NETCONF :private-candidate capability;
         If the server advertises the :private-candidate
         capability for a session, then this feature must
         also be enabled for that session.  Otherwise,
         this feature must not be enabled.";
      reference
        "draft-ietf-netconf-privcand";
    }

    rpc update {
      if-feature "private-candidate";
      description
        "Updates the private candidate from the running
         configuration.";
      reference
        "draft-ietf-netconf-privcand";
      input {
        leaf resolution-mode {
          type enumeration {
            enum revert-on-conflict {
              description
                "Reject update when any conflicting
                 node is found and revert the private
                 candidate configuration datastore to its
                 state prior to issuing the update.";
            }
            enum prefer-candidate {
              description
                "Resolve conflicted node by selecting
                 the private candidate configuration
                 datastore version.";
            }
            enum prefer-running {
              description
                "Resolve conflicted node by selecting
                 the running configuration datastore
                 version.";
            }
```

```
            }
            default "revert-on-conflict";
            description
              "Mode to resolve conflicts between running and
               private-candidate configurations.";
          }
        }
      }
    }
    <CODE ENDS>
```

A.2. ietf-netconf-private-candidate-compare@2026-02-03.yang

```
    <CODE BEGINS>
      file "ietf-netconf-private-candidate-compare@2026-02-03.yang"
    module ietf-netconf-private-candidate-compare {
      yang-version 1.1;
      namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-private-candidate-compar
e";
      prefix pccmp;

      import ietf-netconf-private-candidate {
        prefix pc;
      }
      import ietf-nmda-compare {
        prefix cmp;
      }

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

         Editor:   James Cumming
                <james.cumming@nokia.com>

         Editor:   Robert Wills
                <rowills@cisco.com>";
      description
        "NETCONF private candidate support.

         Copyright (c) 2026 IETF Trust and the persons identified as
         authors of the code.  All rights reserved.

         Redistribution and use in source and binary forms, with or
         without modification, is permitted pursuant to, and subject to
         the license terms contained in, the Revised BSD License set
         forth in Section 4.c of the IETF Trust's Legal Provisions
```

```
        Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

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

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
        'MAY', and 'OPTIONAL' in this document are to be interpreted as
        described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

    revision 2026-02-03 {
      description
        "Introduce private candidate support";
      reference
        "draft-ietf-netconf-privcand:
         Netconf Private Candidates";
    }
    revision 2025-10-30 {
      description
        "Introduce private candidate support";
      reference
        "draft-ietf-netconf-privcand:
         Netconf Private Candidates";
    }
    revision 2024-09-12 {
      description
        "Introduce private candidate support";
      reference
        "draft-ietf-netconf-privcand:
         Netconf Private Candidates";
    }

    augment "/cmp:compare/cmp:input" {
      if-feature "pc:private-candidate";
      description
        "Augment private candidate extensions into the NETCONF compare
         RPC";
      leaf reference-point {
        type enumeration {
          enum last-update {
            description
              "Compare using the point the private
               candidate configuration datastore was
               last updated using the update or commit
               RPCs.";
```

```
          }
          enum creation-point {
            description
              "Compare using the point the private
               candidate configuration datastore was
               initially created.";
          }
        }
        default "last-update";
        description
          "When this leaf is provided and the source or
           destination is the candidate datastore, operating
           in private candidate mode, the comparison will
           either occur between the last-update point of
           the private candidate or the creation-point of
           the private candidate.";
        reference
          "draft-ietf-netconf-privcand";
      }
    }
  }
  <CODE ENDS>
```

Acknowledgements

The authors would like to thank Andy Bierman, Jan Lindblad, Lori-Ann McGrath, Dylan Sadoun, Jason Sterne, Kent Watsen and Rob Wilton for their reviews and feedback.

Authors' Addresses

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

Robert Wills
Cisco Systems
Email: rowills@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 29 December 2026

P. Andersson
Ionio Systems

27 June 2026

YANG Groupings for QUIC clients and QUIC servers
draft-ietf-netconf-quic-client-server-08

Abstract

This document defines five YANG 1.1 modules to support the configuration of QUIC clients and QUIC servers. The modules include basic parameters for configuring QUIC based clients and servers as well as initial modules for the IANA registries "QUIC Versions" and "QUIC Transport Parameters".

Editorial note (To be removed by the RFC Editor)

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

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

* AAAA --> the assigned RFC value for this draft

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 29 December 2026.

Table of Contents

1. Introduction

This documents defines two YANG 1.1 [RFC7950] modules to support the configuration of QUIC clients and QUIC servers (QUIC is defined in [RFC9000]), either as standalone or in conjunction with configuration of other protocol layers.

1.1. Terminology

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

The following terms are defined in [RFC7950] and are not redefined here: client, data model, data tree, feature, extension, module, leaf, leaf-list, and server.

2. The "ietf-quic-common" Module

This section defines a YANG 1.1 module called "ietf-quic-common".

This YANG module has normative references to [RFC9000], [RFC9221], [RFC9287], [RFC9312], [RFC9368], [RFC9369], and [I-D.ietf-netmod-yang-semver].

2.1. Data model overview

This section presents an overview of of the "ietf-quic-common" module in terms of features and groupings.

2.1.1. Groupings

The "ietf-quic-common" module defines the following "grouping" statement:

* transport-parameters

This grouping is presented in the following subsection.

2.1.1.1. The "quic-common" Grouping

The following tree diagram [RFC8340] illustrates the "quic-common" grouping:

```
  grouping version:
    +-- version*   iana-quic-versions:version
  grouping transport-parameters:
    +-- transport-parameter*
            iana-quic-transport:transport-parameter
```

Comments:

* This grouping contains common transport parameters for QUIC connections.

2.2. YANG Module

<CODE BEGINS> file "ietf-quic-common@1.0.0-draft-ietf-netconf-quic-client-server-08.yang"

```
module ietf-quic-common {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-quic-common";
  prefix quiccmn;

  import iana-quic-versions {
    prefix iana-quic-versions;
    reference
      "RFC AAAA: YANG Groupings for QUIC clients and QUIC servers";
  }

  import iana-quic-transport {
    prefix iana-quic-transport;
    reference
      "RFC AAAA: YANG Groupings for QUIC clients and QUIC servers";
  }

  import ietf-yang-semver {
    prefix ysv;
    reference
      "draft-ietf-netmod-yang-semver: YANG Semantic Versioning";
  }


  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG List: NETCONF WG list <mailto:netconf@ietf.org>
     WG Web:  https://datatracker.ietf.org/wg/netconf
     Author:  Per Andersson <mailto:per.ietf@ionio.se>";
```

```
  description
    "This module defines a reusable grouping that is common for
     QUIC clients and QUIC servers. This grouping statement is
     used by both 'ietf-quic-client' and 'ietf-quic-server'
     modules.

     Copyright (c) 2026 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 AAAA
     (https://www.rfc-editor.org/info/rfcAAAA); see the RFC
     itself for full legal notices.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
     'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
     'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
     are to be interpreted as described in BCP 14 (RFC 2119)
     (RFC 8174) when, and only when, they appear in all
     capitals, as shown here.";

  revision 2026-06-27 {
    ysv:version 1.0.0-draft-quic-client-server-latest;
    description
      "Initial version";
    reference
      "RFC AAAA: YANG Groupings for QUIC Clients and QUIC Servers";
  }

  // Groupings

  grouping version {
    description
      "A reusable grouping for QUIC Versions.";
    reference
      "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport
       RFC 9369: QUIC Version 2";

    leaf-list version {
      type iana-quic-versions:version;
      description
        "The QUIC versions supported.";
```

```
      }
    }

    grouping transport-parameters {
      description
        "A reusable grouping for QUIC Transport Parameters.";
      reference
        "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport
         RFC 9221: An Unreliable Datagram Extension to QUIC
         RFC 9287: Greasing the QUIC Bit
         RFC 9312: Manageability of the QUIC Transport Protocol
         RFC 9368: Compatible Version Negotiation for QUIC";

      leaf-list transport-parameter {
        type iana-quic-transport:transport-parameter;
        description
          "A leaf list containing the QUIC transport parameters.";
      }
    }
  }
```

<CODE ENDS>

3. The "ietf-quic-client" Module

This section defines a YANG 1.1 module called "ietf-quic-client".

3.1. Data model overview

This section presents an overview of of the "ietf-quic-client" module in terms of features and groupings.

3.1.1. Features

The module itself does not define any features. However, in order to require TLS 1.3 the following "if-feature" is defined "tlscmn:tls13 not tlscmn:tls12". For QUIC TLS requirements see [RFC9001].

For further details about available features see the "ietf-tls-client" and "ietf-udp-client" modules. defined in [RFC9645] and [RFC9984] respectively.

3.1.2. Groupings

The "ietf-quic-client" module defines the following "grouping" statement:

* quic-client

This grouping is presented in the following subsection.

3.1.2.1. The "quic-client" Grouping

The following tree diagram [RFC8340] illustrates the "quic-client" grouping:

```
  grouping quic-client:
    +---u tlsc:tls-client-grouping
    |       {tlscmn:tls13 and not tlscmn:tls12}?
    +---u udpc:udp-client
    +---u quiccmn:version
    +---u quiccmn:transport-parameters
```

Comments:

* This grouping uses the "tls-client-grouping" grouping discussed in [RFC9645]. Note that QUIC requires TLS 1.3 (or later), thus the "if-feature" invariant "tlscmn:tls13 and not tlscmn:tls12" is defined for this grouping.

* This grouping uses the "udp-client-grouping" grouping discussed in [RFC9984].

3.2. YANG Module

This YANG module has normative references to [RFC9645] and [RFC9984], and [I-D.ietf-netmod-yang-semver].

<CODE BEGINS> file "ietf-quic-client@1.0.0-draft-ietf-netconf-quic-client-server-08.yang"

```
module ietf-quic-client {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-quic-client";
  prefix quicc;

  import ietf-quic-common {
    prefix quiccmn;
    reference
      "RFC AAAA: YANG Groupings for QUIC Clients and QUIC Servers";
  }

  import ietf-tls-client {
    prefix tlsc;
    reference
      "RFC 9645: YANG Groupings for TLS Clients and TLS Servers";
```

```
  }

  import ietf-tls-common {
    prefix tlscmn;
    reference
      "RFC 9645: YANG Groupings for TLS Clients and TLS Servers";
  }

  import ietf-udp-client {
    prefix udpc;
    reference
      "RFC 9984: YANG Groupings for UDP Clients and UDP Servers";
  }

  import ietf-yang-semver {
    prefix ysv;
    reference
      "draft-ietf-netmod-yang-semver: YANG Semantic Versioning";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG List: NETCONF WG list <mailto:netconf@ietf.org>
     WG Web:  https://datatracker.ietf.org/wg/netconf
     Author:  Per Andersson <mailto:per.ietf@ionio.se>";

  description
    "This module defines reusable groupings for QUIC clients that
     can be used as a basis for specific QUIC client instances.

     Copyright (c) 2026 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 AAAA
     (https://www.rfc-editor.org/info/rfcAAAA); see the RFC
     itself for full legal notices.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
     'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
```

```
        'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
        are to be interpreted as described in BCP 14 (RFC 2119)
        (RFC 8174) when, and only when, they appear in all
        capitals, as shown here.";

    revision 2026-06-27 {
      ysv:version 1.0.0-draft-ietf-netconf-quic-client-server-08;
      description
        "Initial version";
      reference
        "RFC AAAA: YANG Groupings for QUIC Clients and QUIC Servers";
    }

    // Groupings

    grouping quic-client {
      description
        "Grouping to configure a QUIC client.";
      reference
        "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport";

      uses tlsc:tls-client-grouping {
        if-feature "tlscmn:tls13 and not tlscmn:tls12";
        description
          "QUIC requires that TLS 1.3 (or later) is used.";
        reference
          "RFC 9001: Using TLS to Secure QUIC";
      }
      uses udpc:udp-client;
      uses quiccmn:version;
      uses quiccmn:transport-parameters;
    }
  }

  <CODE ENDS>
```

4. The "ietf-quic-server" Module

This section defines a YANG 1.1 module called "ietf-quic-server".

4.1. Data model overview

This section presents an overview of of the "ietf-quic-server" module in terms of features and groupings.

4.1.1. Features

The module itself does not define any features. However, in order to require TLS 1.3 the following "if-feature" is defined "tlscmn:tls13 not tlscmn:tls12". For QUIC TLS requirements see [RFC9001].

For further details about available features see the "ietf-tls-server" and "ietf-udp-server" modules, defined in [RFC9645] and [RFC9984] respectively.

4.1.2. Groupings

The "ietf-quic-server" module defines the following "grouping" statement:

* quic-server

This grouping is presented in the following subsection.

4.1.2.1. The "quic-server" Grouping

The following tree diagram [RFC8340] illustrates the "quic-server" grouping:

```
  grouping quic-server:
    +---u tlss:tls-server-grouping
    |       {tlscmn:tls13 and not tlscmn:tls12}?
    +---u udps:udp-server
    +---u quiccmn:version
    +---u quiccmn:transport-parameters
```

Comments:

* This grouping uses the "tls-server-grouping" grouping discussed in [RFC9645]. Note that QUIC requires TLS 1.3 (or later), thus the "if-feature" invariant "tlscmn:tls13 and not tlscmn:tls12" is defined for this grouping.

* This grouping uses the "udp-server-grouping" grouping discussed in [RFC9984].

4.2. YANG Module

This YANG module has normative references to [RFC9645], [RFC9984], and [I-D.ietf-netmod-yang-semver].

<CODE BEGINS> file "ietf-quic-server@1.0.0-draft-ietf-netconf-quic-client-server-08.yang"

```
   module ietf-quic-server {
     yang-version 1.1;
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-quic-server";
     prefix quics;

     import ietf-quic-common {
       prefix quiccmn;
       reference
         "RFC AAAA: YANG Groupings for QUIC Clients and QUIC Servers";
     }

     import ietf-tls-server {
       prefix tlss;
       reference
         "RFC 9645: YANG Groupings for TLS Clients and TLS Servers";
     }

     import ietf-tls-common {
       prefix tlscmn;
       reference
         "RFC 9645: YANG Groupings for TLS Clients and TLS Servers";
     }

     import ietf-udp-server {
       prefix udps;
       reference
         "RFC 9984: YANG Groupings for UDP Clients and UDP Servers";
     }

     import ietf-yang-semver {
       prefix ysv;
       reference
         "draft-ietf-netmod-yang-semver: YANG Semantic Versioning";
     }

     organization
       "IETF NETCONF (Network Configuration) Working Group";

     contact
       "WG List: NETCONF WG list <mailto:netconf@ietf.org>
        WG Web:  https://datatracker.ietf.org/wg/netconf
        Author:  Per Andersson <mailto:per.ietf@ionio.se>";

     description
       "This module defines reusable groupings for QUIC servers that
        can be used as a basis for specific QUIC server instances.
```

```
        Copyright (c) 2026 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 AAAA
        (https://www.rfc-editor.org/info/rfcAAAA); see the RFC
        itself for full legal notices.

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
        'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
        'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
        are to be interpreted as described in BCP 14 (RFC 2119)
        (RFC 8174) when, and only when, they appear in all
        capitals, as shown here.";

    revision 2026-06-27 {
      ysv:version 1.0.0-draft-quic-client-server-latest;
      description
        "Initial version";
      reference
        "RFC AAAA: YANG Groupings for QUIC Clients and QUIC Servers";
    }

    // Groupings

    grouping quic-server {
      description
        "Grouping to configure a QUIC server.";
      reference
        "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport";

      uses tlss:tls-server-grouping {
        if-feature "tlscmn:tls13 and not tlscmn:tls12";
        description
          "QUIC requires that TLS 1.3 (or later) is used.";
        reference
          "RFC 9001: Using TLS to Secure QUIC";
      }
      uses udps:udp-server;
      uses quiccmn:version;
      uses quiccmn:transport-parameters;
    }
```

```
}

<CODE ENDS>
```

5. Operational Considerations

This section is modeled after Section 3 of [I-D.ietf-opsawg-rfc5706bis].

When using the groupings defined in this document, it is possible to refine the leaves and add default values. Such default values will be used as default settings when establishing connections.

Updating settings for QUIC connections is implementation specific behavior and dependent. Changing settings could be done for existing connections if such support is available in the QUIC library used.

Updating settings in real time for established QUIC connections may have major impact on the connection, including early termination. Moreover, the client configuring the settings may not know the various internal states of the connection, which may further bring unexpected consequences. Just as one example, changing the max_idle_timeout to a too low value would possibly close the connection. Implementations should be carefully support changing established connections parameters, as well as users changing those parameters.

6. Security Considerations

This section is modeled after the template described in Section 3.7.1 of [RFC9907].

The "ietf-quic-common", "ietf-quic-client", and "ietf-quic-server" YANG modules defines a data model that is designed to be accessed via YANG-based management protocols, such as the Network Configuration Protocol (NETCONF) [RFC6241] and RESTCONF [RFC8040]. These YANG-based management protocols (1) have to use a secure transport layer (e.g., Secure Shell (SSH) [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and (2) 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 preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The YANG module defines a set of identities, types, and groupings. These nodes are intended to be reused by other YANG modules. The module by itself does not expose any data nodes that are writable,

data nodes that contain read-only state, or RPCs. As such, there are no additional security issues related to the YANG module that need to be considered.

Modules that use the groupings that are defined in this document should identify the corresponding security considerations. For example, reusing some of these groupings will expose privacy-related information. Security considerations for the groupings used in the modules are discussed in [RFC9645] and [RFC9984], refer to these documents for further details.

7. IANA Considerations

7.1. The "IETF XML" Registry

This document registers five URIs in the "ns" subregistry of the IETF XML Registry [RFC3688] maintained at https://www.iana.org/assignments/xml-registry/xml-registry.xhtml#ns. Following the format in [RFC3688], the following registration is requested:

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

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

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

URI: urn:ietf:params:xml:ns:yang:iana-quic-versions
Registrant Contact: IANA
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:iana-quic-transport
Registrant Contact: IANA
XML: N/A, the requested URI is an XML namespace.

7.2. The "YANG Module Names" Registry

This document registers five YANG modules in the YANG Module Names registry [RFC6020] maintained at https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml. Following the format defined in [RFC6020], the below registration is requested:

```
   name: ietf-quic-common
   namespace: urn:ietf:params:xml:ns:yang:ietf-quic-common
   prefix: quiccmn
   RFC: AAAA

   name: ietf-quic-client
   namespace: urn:ietf:params:xml:ns:yang:ietf-quic-client
   prefix: quicc
   RFC: AAAA

   name: ietf-quic-server
   namespace: urn:ietf:params:xml:ns:yang:ietf-quic-server
   prefix: quics
   RFC: AAAA

   name: iana-quic-versions
   namespace: urn:ietf:params:xml:ns:yang:iana-quic-versions
   prefix: iana-quic-versions
   RFC: AAAA

   name: iana-quic-transport
   namespace: urn:ietf:params:xml:ns:yang:iana-quic-transport
   prefix: iana-quic-transport
   RFC: AAAA
```

7.3. Considerations for IANA Maintained Modules

The initial YANG modules are created with the iana-yang tool from <https://github.com/llhotka/iana-yang>.

7.3.1. The "iana-quic-versions" Module

IANA is requested to maintain a YANG module called "iana-quic-versions" that shadows the "QUIC Versions" registry [IANA-QUIC-VERSIONS].

Ths registry defines a YANG enumeration for each QUIC version.

An initial version of this module can be found in Appendix A.1.

* Note that this module was created on July 6th, 2025, and that additional entries may have been added before this document's publication. If this is the case, IANA may publish an updated module containing the new entries, or publish the initial module with a "revision" containing the additional QUIC versions.

7.3.2. The "iana-quic-transport" Module

IANA is requested to maintain a YANG module called "iana-quic-transport" that shadows the "QUIC Transport Parameters" registry [IANA-QUIC-TRANSPORT].

Ths registry defines a YANG enumeration for each QUIC transport parameter.

An initial version of this module can be found in Appendix A.2.

Since the QUIC Transport Parameters registry may have provisionally registered names change value when they are made permanent, make the following considerations.

* If an existing name exists in the module, update the value in the description field.

* Since normative changes to enum descriptions are non-backwards-compatible, the YANG Semver version need to update the MAJOR version number. Note that this follows normal YANG Semver versioning rules. [I-D.ietf-netmod-yang-semver].

* Note that this module was created on July 6th, 2025, and that additional entries may have been added before this document's publication. If this is the case, IANA may publish an updated module containing the new entries, or publish the initial module with a "revision" containing the additional QUIC transport parameters.

8. References

8.1. Normative References

[I-D.ietf-netmod-yang-module-filename]
Andersson, P., "YANG module file name convention", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-filename-14, 4 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-filename-14>.

[I-D.ietf-netmod-yang-semver]
Clarke, J., Wilton, R., Rahman, R., Lengyel, B., Sterne, J., and B. Claise, "YANG Semantic Versioning", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-semver-26, 7 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-26>.

[I-D.ietf-opsawg-rfc5706bis]
Claise, B., Clarke, J., Farrel, A., Barguil, S., Pignataro, C., and R. Chen, "Guidelines for Considering Operations and Management in IETF Specifications", Work in Progress, Internet-Draft, draft-ietf-opsawg-rfc5706bis-05, 26 June 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-rfc5706bis-05>.

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

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

[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/info/rfc4252>.

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

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

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

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

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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/info/rfc9000>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>.

[RFC9221] Pauly, T., Kinnear, E., and D. Schinazi, "An Unreliable Datagram Extension to QUIC", RFC 9221, DOI 10.17487/RFC9221, March 2022, <https://www.rfc-editor.org/info/rfc9221>.

[RFC9287] Thomson, M., "Greasing the QUIC Bit", RFC 9287, DOI 10.17487/RFC9287, August 2022, <https://www.rfc-editor.org/info/rfc9287>.

[RFC9312] KÃ¼hlewind, M. and B. Trammell, "Manageability of the QUIC Transport Protocol", RFC 9312, DOI 10.17487/RFC9312, September 2022, <https://www.rfc-editor.org/info/rfc9312>.

[RFC9368] Schinazi, D. and E. Rescorla, "Compatible Version Negotiation for QUIC", RFC 9368, DOI 10.17487/RFC9368, May 2023, <https://www.rfc-editor.org/info/rfc9368>.

[RFC9369] Duke, M., "QUIC Version 2", RFC 9369, DOI 10.17487/RFC9369, May 2023, <https://www.rfc-editor.org/info/rfc9369>.

[RFC9645] Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", RFC 9645, DOI 10.17487/RFC9645, October 2024, <https://www.rfc-editor.org/info/rfc9645>.

[RFC9907] Bierman, A., Boucadair, M., Ed., and Q. Wu, "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 9907, DOI 10.17487/RFC9907, March 2026, <https://www.rfc-editor.org/info/rfc9907>.

[RFC9984] Huang-Feng, A., Francois, P., and K. Watsen, "YANG Groupings for UDP Clients and UDP Servers", RFC 9984, DOI 10.17487/RFC9984, June 2026, <https://www.rfc-editor.org/info/rfc9984>.

8.2. Informative References

[IANA-QUIC-TRANSPORT]
(IANA), I. A. N. A., "QUIC Transport Parameters", <https://www.iana.org/assignments/quic/quic.xhtml#quic-transport>.

[IANA-QUIC-VERSIONS]
(IANA), I. A. N. A., "QUIC Versions", <https://www.iana.org/assignments/quic/quic.xhtml#quic-versions>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. YANG Modules for IANA

The initial YANG modules are created with the iana-yang tool from <https://github.com/llhotka/iana-yang>.

A.1. Initial Module for the "QUIC Versions" Registry

This YANG module has normative references to [RFC9000] and [RFC9369].

<CODE BEGINS> file "iana-quic-versions@2026-05-04.yang"

```
module iana-quic-versions {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:iana-quic-versions";
  prefix iana-quic-versions;

  organization
    "Internet Assigned Numbers Authority (IANA)";

  contact
    "Internet Assigned Numbers Authority

     ICANN
     12025 Waterfront Drive, Suite 300
     Los Angeles, CA 90094

     Tel: +1 424 254 5300

     <mailto:iana@iana.org>";
```

```
    description
      "This YANG module translates IANA registry 'QUIC Versions' to
       YANG derived types.

       Copyright (c) 2026 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 was generated from the
       corresponding IANA registry using an XSLT stylesheet from the
       'iana-yang' project (https://github.com/llhotka/iana-yang).";

    reference
      "QUIC (https://www.iana.org/assignments/quic/)";

    revision 2026-05-04 {
      description
        "Current revision as of the revision date specified in the XML
         representation of the registry page.";
      reference
        "https://www.iana.org/assignments/quic/quic.xml";
    }

    /* Typedefs */

    typedef quic-version {
      type enumeration {
        enum 0x00000000 {
          description
            "(permanent) Reserved for Version Negotiation";
          reference
            "RFC 9000";
        }
        enum 0x00000001 {
          description
            "(permanent)";
          reference
            "RFC 9000";
        }
        enum 0x51303433 {
          description
            "(provisional) Google QUIC Q043";
```

```
        }
        enum 0x51303436 {
          description
            "(provisional) Google QUIC Q046";
        }
        enum 0x51303530 {
          description
            "(provisional) Google QUIC Q050";
        }
        enum 0x6b3343cf {
          description
            "(permanent)";
          reference
            "RFC 9369";
        }
        enum 0x6f7dc0fd {
          description
            "(provisional) SCONE Protocol - Even Signal Values";
          reference
            "draft-ietf-scone-protocol-04";
        }
        enum 0x709a50c4 {
          description
            "(provisional) QUIC v2 draft codepoint";
          reference
            "RFC 9369";
        }
        enum 0xef7dc0fd {
          description
            "(provisional) SCONE Protocol - Odd Signal Values";
          reference
            "draft-ietf-scone-protocol-04";
        }
      }
      description
        "This enumeration type defines QUIC protocol versions.";
      reference
        "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure
         Transport";
    }

    typedef version {
      type quic-version;
      description
        "This type allows reference to a QUIC version using the
         assigned value.";
    }
  }
```

```
   <CODE ENDS>
```

A.2. Initial Module for the "QUIC Transport Parameters" Registry

This YANG module has normative references to [RFC9000], [RFC9221], [RFC9287], [RFC9312], and [RFC9368],

```
   <CODE BEGINS> file "iana-quic-transport@2026-05-04.yang"

   module iana-quic-transport {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:iana-quic-transport";
     prefix iana-quic-transport;

     organization
       "Internet Assigned Numbers Authority (IANA)";

     contact
       "Internet Assigned Numbers Authority

        ICANN
        12025 Waterfront Drive, Suite 300
        Los Angeles, CA 90094

        Tel: +1 424 254 5300

        <mailto:iana@iana.org>";

     description
       "This YANG module translates IANA registry 'QUIC Transport
        Parameters' to YANG derived types.

        Copyright (c) 2026 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 was generated from the
        corresponding IANA registry using an XSLT stylesheet from the
        'iana-yang' project (https://github.com/llhotka/iana-yang).";

     reference
       "QUIC (https://www.iana.org/assignments/quic/)";
```

```
    revision 2026-05-04 {
      description
        "Current revision as of the revision date specified in the XML
         representation of the registry page.";
      reference
        "https://www.iana.org/assignments/quic/quic.xml";
    }

    /* Typedefs */

    typedef quic-transport-parameter {
      type enumeration {
        enum original_destination_connection_id {
          description
            "0x00 (permanent)";
          reference
            "RFC 9000";
        }
        enum max_idle_timeout {
          description
            "0x01 (permanent)";
          reference
            "RFC 9000";
        }
        enum stateless_reset_token {
          description
            "0x02 (permanent)";
          reference
            "RFC 9000";
        }
        enum max_udp_payload_size {
          description
            "0x03 (permanent)";
          reference
            "RFC 9000";
        }
        enum initial_max_data {
          description
            "0x04 (permanent)";
          reference
            "RFC 9000";
        }
        enum initial_max_stream_data_bidi_local {
          description
            "0x05 (permanent)";
          reference
            "RFC 9000";
        }
```

```
        enum initial_max_stream_data_bidi_remote {
          description
            "0x06 (permanent)";
          reference
            "RFC 9000";
        }
        enum initial_max_stream_data_uni {
          description
            "0x07 (permanent)";
          reference
            "RFC 9000";
        }
        enum initial_max_streams_bidi {
          description
            "0x08 (permanent)";
          reference
            "RFC 9000";
        }
        enum initial_max_streams_uni {
          description
            "0x09 (permanent)";
          reference
            "RFC 9000";
        }
        enum ack_delay_exponent {
          description
            "0x0a (permanent)";
          reference
            "RFC 9000";
        }
        enum max_ack_delay {
          description
            "0x0b (permanent)";
          reference
            "RFC 9000";
        }
        enum disable_active_migration {
          description
            "0x0c (permanent)";
          reference
            "RFC 9000";
        }
        enum preferred_address {
          description
            "0x0d (permanent)";
          reference
            "RFC 9000";
        }
```

```
        enum active_connection_id_limit {
          description
            "0x0e (permanent)";
          reference
            "RFC 9000";
        }
        enum initial_source_connection_id {
          description
            "0x0f (permanent)";
          reference
            "RFC 9000";
        }
        enum retry_source_connection_id {
          description
            "0x10 (permanent)";
          reference
            "RFC 9000";
        }
        enum version_information {
          description
            "0x11 (permanent)";
          reference
            "RFC 9368";
        }
        enum max_datagram_frame_size {
          description
            "0x20 (permanent)";
          reference
            "RFC 9221";
        }
        enum initial_max_path_id {
          description
            "0x3e (permanent)";
          reference
            "RFC-ietf-quic-multipath-21";
        }
        enum discard {
          description
            "0x173e (provisional) Receiver silently discards.";
          reference
            "https://github.com/quicwg/base-drafts/wiki
             /Quantum-Readiness-test";
        }
        enum scone_supported {
          description
            "0x219e (provisional)";
          reference
            "draft-ietf-scone-protocol-04";
```

```
        }
        enum google_handshake_message {
          description
            "0x26ab (provisional) Used to carry Google internal
             handshake message";
        }
        enum grease_quic_bit {
          description
            "0x2ab2 (permanent)";
          reference
            "RFC 9287";
        }
        enum initial_rtt {
          description
            "0x3127 (provisional) Initial RTT in microseconds";
        }
        enum google_connection_options {
          description
            "0x3128 (provisional) Google connection options for
             experimentation";
        }
        enum user_agent {
          status deprecated;
          description
            "0x3129 (provisional) User agent string (deprecated)";
        }
        enum google_version {
          status deprecated;
          description
            "0x4752 (provisional) Deprecated; use version_information
             instead";
        }
        enum version_information_draft {
          status deprecated;
          description
            "0xff73db (provisional) Deprecated; use version_information
             instead";
          reference
            "draft-ietf-quic-version-negotiation-13";
        }
        enum google_debug_1 {
          description
            "0x219bbcd0 (provisional)";
        }
        enum min_ack_delay {
          description
            "0xff04de1b (provisional)";
          reference
```

```
            "draft-ietf-quic-ack-frequency-07";
        }
        enum bdp_frame {
          description
            "0x4143414213370002 (provisional)";
          reference
            "draft-misell-quic-bdp-token-02";
        }
      }
      description
        "This enumeration type defines QUIC transport parameters.";
      reference
        "RFC 9000: QUIC: A UDP-Based Multiplexed and Secure
         Transport";
    }

    typedef transport-parameter {
      type quic-transport-parameter;
      description
        "This type allows reference to a QUIC transport parameters
         using the assigned value.";
    }
  }
```

<CODE ENDS>

Acknowledgements

The author would like to thank Marc Blanchet for his excellent technical review, support, and guidance.

Author's Address

Per Andersson
Ionio Systems
Email: per.ietf@ionio.se

Network Configuration
Internet-Draft
Intended status: Standards Track
Expires: 25 December 2026

R. Wilton, Ed.
Cisco Systems
H. Keller
Deuetsche Telekom
B. Claise
Everything OPS
E. Aries
Juniper
J. Cumming
Nokia
T. Graf
Swisscom

23 June 2026

YANG Datastore Telemetry (YANG Push version 2)
draft-ietf-netconf-yang-push-2-00

Abstract

YANG Push version 2 is a YANG datastore telemetry solution, as an alternative lightweight 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-ietf-netconf-yang-push-2/.

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.

Copyright Notice

Table of Contents

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 v2 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 v2", an lightweight alternative to Subscribed Notifications [RFC8639] and YANG Push [RFC8641]. YANG Push v2 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 v2 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 v2 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 v2

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 is 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 v2.

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 allows a single subscription to be configured as both periodic and on-change.

3.4. Relationships to existing RFCs and Internet Drafts

This document, specifying YANG Push v2, 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 v2 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 v2 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 v2 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-yang-push-2 YANG module, to augment YANG Push v2 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-yang-push-2 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 v2 solution anticipates and requires new bis 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-yang-push-2 data model.

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

This document reuses the work from [I-D.draft-ietf-netconf-distributed-notif], but with some changes to work with the YANG Push v2 data models. This is described in Section 12.

4. YANG Push v2 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 v2 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 Section 7.3.

* a choice of encoding of the messages, e.g., JSON, or CBOR.

* a receiver to 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 the receiver 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 receiver, 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 16.

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 13.2 and associated tree view in Section 13.1. 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 the receiver wishes 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*

This document also uses the terminology from [I-D.ietf-netconf-distributed-notif] in Section 12 and the related examples in Appendix C.5.

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.

Filters can either be defined inline within a configured subscription (Figure 21), 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 v2 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.

* _XPaths_: 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 13.

```
module: ietf-yang-push-2-config
  +--rw datastore-telemetry!
     +--rw filters
        +--rw filter* [name]
           +--rw name              string
           +--rw (filter)
              +--:(path)
              |  +--rw path        ypath
              +--:(subtree)
              |  +--rw subtree     <anydata>
              +--:(xpath)
                 +--rw xpath       yang:xpath1.0 {yp2:xpath}?

  augment /yp2:subscription-started/yp2:target:
    +-- filter-ref?   filter-ref
  augment /yp2:subscription-modified/yp2: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 v2 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 v2, 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 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 enqueuing 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 9.2.3).

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 _publisher-id_ MAY be excluded in it matches the default value (0), otherwise it MUST be included.

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.

The _event-time_ field is used to correlate separate _update_ messages that collectively represent individual parts of the same update (e.g., where the source data is being obtained from multiple producers).

```
{
  "ietf-yp-notification:envelope": {
    "event-time": "2024-10-10T08:00:05.22Z",
    "hostname": "example-router",
    "sequence-number": 3219,
    "contents": {
      "ietf-yang-push-2:update": {
        "id": 1011,
        "path-prefix": "/ietf-interfaces:interfaces",
        "snapshot-type": "periodic",
        "observation-time": "2024-10-10T08:00:05.11Z",
        "updates": [
          {
            "target-path": "interface",
            "replace-by": {
              "interface": [
                {
                  "name": "eth0",
                  "type": "iana-if-type:ethernetCsmacd",
                  "enabled": true,
                  "oper-status": "up",
                  "admin-status": "up"
                },
                {
                  "name": "eth1",
                  "type": "iana-if-type:ethernetCsmacd",
                  "enabled": true,
                  "oper-status": "up",
                  "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)?
|  |     +--:(merge)
|  |     |  +--ro merge?         <anydata>
|  |     +--:(replaced-by)
|  |     |  +--ro replaced-by?   <anydata>
|  |     +--:(deleted)
|  |        +--ro deleted?       empty
|  +--ro complete?           boolean
```

Figure 3: 'update' notification

The normative definitions for the notifications fields are given in the YANG module in Section 13. 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 by the _merge_, _replace-by_, or _deleted_ data.

* _data_ - A choice representing the action taken and the updated data, which is one of the following:

  - _merge_ - identifies the data node that the receiver should merge with their current view of that data node. I.e., all descendant data nodes reporting values should replace those held by the receiver, but the absence of a decendent data node does not imply that it no longer exists and hence should be deleted.

  - _replaced-by_ - identifies the data node that replaces the copy of that data node in the receiver's state. Any descendent data nodes not present in the update no longer exist and hence should be implicitly deleted in the receiver's current state.

  - _deleted_ - indicates that the data node no longer exists and should be deleted in the receivers state. This field _MUST NOT_ be sent in an _on-change_ update message.

* _complete_ - if present, this flag indicates whether the notification is complete or whether more messages as part of the same update will follow. The default value for the flag depends on the _snapshot-type_. For _on-change_ events, the messages are assumed to be completed until the _complete_ leaf is set to false. For other type of events (e.g., periodic), the messages are assumed to be incomplete unless the _complete_ leaf is set to true. Setting this flag to true 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-yang-push-2-config
  +--rw datastore-telemetry!
     +--rw subscriptions
        +--rw subscription* [id]
           +--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 /yp2:subscription-started/yp2:target:
    +-- filter-ref?   filter-ref
  augment /yp2:subscription-modified/yp2:target:
    +-- filter-ref?   filter-ref
```

Figure 4: 'update-trigger' container

The normative definitions for the update-trigger fields are given in the _ietf-yang-push-2_ YANG module in Section 13. 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.

Where possible, publishers SHOULD use the _replaced-by_ data node to represent the new data since it provides clients with simple explicit semantics. Publishers MAY use the _merge_ data node when they cannot determine whether the data being published in the update is complete.

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. In the case that data nodes have been changed then the _update_ record SHOULD only report the state that has changed (included any required list keys), but MAY include additional unchanged data nodes if the publisher is unable to optimize the on-change update message.

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 is 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. For each data node covered by an on-change with sync-on-start subscription, then an _sync-on-start_ _update_ notification containing the current state MUST be sent before any

on-change _update_ notifications for those same data nodes. However, _sync-on-start_ and _on-change_ _update_ notifications may be interleaved for different data-nodes under the subscription. Unsolicited _sync-on-start update_ notifications MUST NOT be sent, they MUST only be sent after a subscription has started.

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 MUST be 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 v2?*

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 keep the subscription in _inactive_ state.

7.6. Combined periodic and on-change subscriptions

A single subscription may 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

Figure 5 provides an example of a notification message for a subscription tracking the operational status of interface (per [RFC8343]). This notification message is encoded using JSON [RFC7951].

```
{
  "ietf-yang-push-2:update": {
    "id": "interfaces",
    "path-prefix": "/ietf-interfaces:interfaces/interface",
    "snapshot-type": "periodic",
    "observation-time": "2024-10-10T08:00:05.11Z",
    "updates": [
      {
        "target-path": "",
        "merge": {
          "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": false
  }
}
```

Figure 5: Example periodic update message

Figure 6 provides an example of an on-change notification message for the same subscription.

```
{
  "ietf-yang-push-2:update": {
    "id": "interfaces",
    "path-prefix": "/ietf-interfaces:interfaces",
    "snapshot-type": "on-change-update",
    "observation-time": "2025-10-10T08:16:06.11Z",
    "updates": [
      {
        "target-path": "interface[name='eth0']",
        "replaced-by": {
          "name": "eth0",
          "type": "iana-if-type:ethernetCsmacd",
          "enabled": false,
          "ietf-interfaces:oper-status": "down"
        }
      },
      {
        "target-path": "interface[name='eth1']",
        "replaced-by": {
          "name": "eth1",
          "type": "iana-if-type:ethernetCsmacd",
          "enabled": false,
          "ietf-interfaces:oper-status": "down"
        }
      }
    ]
  }
}
```

Figure 6: Example on-change update message

8. Receivers, Transports, and Encodings

8.1. Receivers

Every subscription is associated with a receiver, which identifies 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. Receivers for Configured Subscriptions

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

Below is a tree diagram for _datastore-telemetry/receivers_ container. All objects contained in this tree are described in the YANG module in Section 13.2.

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-yang-push-2-config
  +--rw datastore-telemetry!
     +--rw receivers
        +--rw receiver* [name]
           +--rw name                        string
           +--rw encoding                    yp2: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 /yp2:subscription-started/yp2:target:
    +-- filter-ref?   filter-ref
  augment /yp2:subscription-modified/yp2: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 13.2 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 v2 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 v2 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:

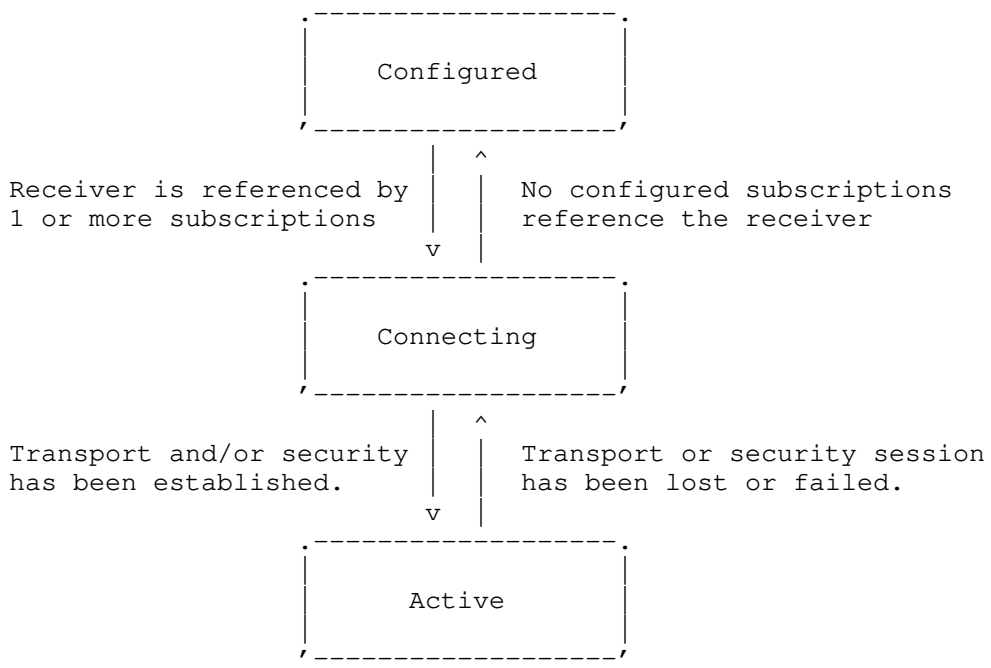


Figure 8: Receiver Session State Diagram

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 v2. The requirements for these transport specifications are documented in the following section:

8.2.1. Requirements for YANG Push v2 Transport Specifications

This section provides requirements for any transport specifications supporting the YANG Push v2 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 14, 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-yang-push-2-capabilities YANG Model Section 15.2.

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 v2 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 v2 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 (using regular YANG identifiers). 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-yang-push-2.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-yang-push-2-capabilities YANG module's transport capabilities Section 15.2.

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:

* an _id_ 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 ids 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 subscription id that conflicts with any other subscription id (configured or dynamic) MUST fail,

* configuring a subscription, assuming it passes configuration validation, replaces any dynamic subscriptions with the same subscription id. Thus, causing the dynamic subscription to be immediately terminated (see Section 9.1.4).

* subscription ids starting with dyn- are reserved for the publisher to use for automatically allocate subscription ids for dynamic subscriptions when the client has choosen not to provide one in the _establish-subscription_ RPC.

9.1.1. Subscription States

YANG Push v2 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 receiver. 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 the receiver yet, and hence has not sent a _subscription-started_ notification. Transport security failures would be in this state.

* *Active*: a valid subscription, connected to the receiver, that has sent a _subscription-stated_ notification and is generating _update_ notifications, as per the terms of the subscription update policy.

* *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.

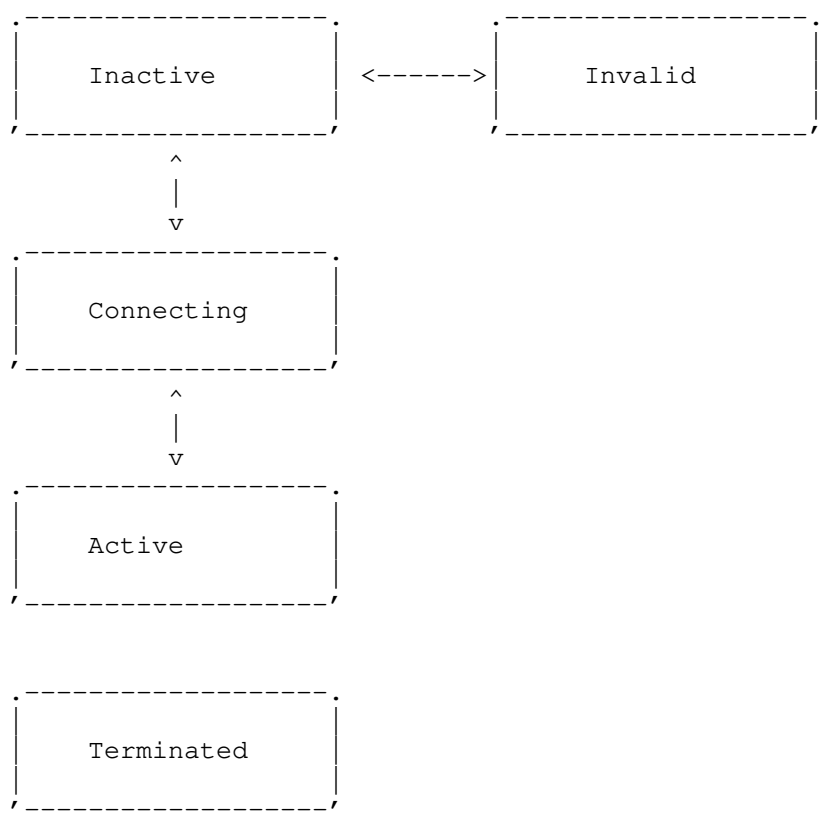


Figure 9: Publisher's States for a Subscription

9.1.2. Creating Subscriptions

After a subscription is successfully established, the publisher immediately sends a _subscription-started_ subscription state change notification to the 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.

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. *TODO, do we always want this behaviour or is it transport specific?*

9.1.3. Modifying Subscriptions

The parameters associated with a subscription MAY be modified by client _modify-subscription_ RPC or through configuration.

If the subscription is in _Active_ state, and hence a _subscription-started_ notification has been enqueued to the receiver, then any subscription parameter changes are handled as per the following sub-sections. If the subscription is not yet in _Active_ state then any transport changes associated with the receiver must be made, but otherwise the new parameters would be notified in the _subscription-started_ notification.

9.1.3.1. Modifications requiring subscription-terminated notification

Changes to any of following parameters MUST terminate the subscription, as per Section 9.1.4, before recreating it, clearing and reinitializing any associated statistics, as per Section 9.1.2:

1. the subcription _id_

2. the _encoding_

3. any _receiver_ settings that change the encoding, transport, transport security, or receiver destination address/port

4. the update-trigger to enable _sync-on-start_.

5. if the _sync-in-start option_ is enabled, then any changes to the subscription-filter (inline or referenced) or YANG schema (_schema-id_) associated with the subscription. _TODO, Why? Should a subscription-modifies message resend the sync-on-start data anyway?_

The _subscription-terminated_ notification MUST be sent using the old _encoding_ and _receiver_ settings before the subscription parameters were changed. The _subscription-started_ notification MUST be sent using the updated subscription parameters.

9.1.3.2. Modifications allowing subscription-modified notification

If changes to a subscription only include changes to the following parameters then they SHOULD be handled via a _subscription-modified_ notification, but MAY be handled as described above. This applies for changes to:

1. the subscription target _filter_ (inline, referring to a different named filter, or changing the referenced filter).

2. the YANG schema (_schema-id_) associated with subscription target filter,

3. the _update-trigger_, unless _sync-on-start_ is enabled.

4. the _description_ field,

5. any other fields that are included in a _subscription-started_ notification message, unless the definition of those fields explicitly specifies different behavior.

9.1.3.3. Modifications requiring no lifecycle notification

Changes to any of the following subscription parameters do not need to be notified to the client:

1. _dscp_ settings.

2. _source-address_, _source-interface_, _source-vrf_, or the source port.

3. any other settings not reported in the _subscription-started_ notification message.

9.1.4. Terminating Subscriptions

Subscriptions MUST 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 the receiver has been lost, either due to network issues, or a failure in the security session state.

5. The publisher does not 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, i.e., as defined in Section 9.1.3.1, that needs the subscription to be reset by terminating and recreating it.

7. The _reset_ RPC is invoked on a configured subscription, or on the referenced receiver associated with a configured subscription.

In addition, from a receiver's perspective, if transport connectivity is lost, then that is equivalent to terminating the subscription via a _subscription-terminated_ notification.

If possible, the publisher MUST try and close the subscription gracefully by generating a _subscription-terminated_ notification to the receiver 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 is 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_ 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 steam 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 is sent for any of these reasons:

1. A new subscription (configured or dynamic) has been started.

2. The properties of a configured subscription has been changed, i.e., as specified in Section 9.1.3.1, that requires a _subscription-terminated_ notification to be sent followed by a _subscription-started_ notification, presuming that the new subscription parameters are valid.

3. A configured subscription previously failed, and was terminated. After the publisher has successfully re-established a connection to the receiver and is ready 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 13.2.

```
+---n subscription-started
|  +--ro id                     subscription-id
|  +--ro description?           string
|  +--ro target
|  |  +--ro datastore?                  identityref
|  |  +--ro (filter)
|  |  |  +--:(path)
|  |  |  |  +--ro path                  ypath
|  |  |  +--:(subtree)
|  |  |  |  +--ro subtree               <anydata>
|  |  |  +--:(xpath)
|  |  |     +--ro xpath                 yang:xpath1.0 {yp2:xpath}?
|  |  +--ro schema-id?                  string
|  |  +--ro yang-library-content-id?
|  |            -> /yanglib:yang-library/content-id
|  +--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
|  +--ro message-publishers
|     +--ro publisher-id*   uint32
```

Figure 10: subscription-started Notification Tree Diagram

9.2.2. "subscription-modified"

The _subscription-modified_ notification is sent to a receiver to indicate that some parameters associated with an active subscription have changed, as per Section 9.1.3.2.

Below is the tree diagram for the _subscription-modified_ notification. Other than the notification name and the _reason_ leaf, the parameters for a _subscription-modified_ notification are the same as for the _subscription-started_ notification. Robust receivers are expected to handle _subscription-started_ and _subscription-modified_ notifications equivalently.

All data nodes contained in this tree diagram are described in the YANG module in Section 13.2.

```
+---n subscription-modified
|  +--ro id                     subscription-id
|  +--ro description?           string
|  +--ro target
|  |  +--ro datastore?                  identityref
|  |  +--ro (filter)
|  |  |  +--:(path)
|  |  |  |  +--ro path                  ypath
|  |  |  +--:(subtree)
|  |  |  |  +--ro subtree               <anydata>
|  |  |  +--:(xpath)
|  |  |     +--ro xpath                 yang:xpath1.0 {yp2:xpath}?
|  |  +--ro schema-id?                  string
|  |  +--ro yang-library-content-id?
|  |            -> /yanglib:yang-library/content-id
|  +--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
|  +--ro message-publishers
|  |  +--ro publisher-id*   uint32
|  +--ro reason                 subscription-change
```

Figure 11: subscription-modified Notification Tree Diagram

9.2.3. "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 subscription has been stopped, either due to the change/removal of some configuration, or an RPC has been invoked to delete or kill a dynamic subscription.

2. The properties of a subscription have been changed, i.e., as specified in Section 9.1.3.1, that requires a _subscription-terminated_ notification to be sent followed by a _subscription-started_ notification, presuming that the new subscription parameters are valid.

3. 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 13.2.

```
+---n subscription-terminated
|  +--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
```

Figure 12: 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.**

9.2.4. "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-yang-push-2-config YANG module Section 14.2 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:

* the receiver for the subscription, as described in Section 8.1. The referenced receiver specifies 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 subscription may reference a filter rather than define it inline as part of the subscription. Even for a referenced filter, the _subscription-started_ and _subscription-modified_ notifications always include the filter specification inline in the notification.

* 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 the publisher detects that 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.

* The lifetime of any transport session(s) to receiver(s) are also tied to the subscription configuration, but in a way that depends on the behavior of the Yang Push 2 transport specificiation, i.e.,

  - For YANG Push transports that specify a separate transport session for each subscription to the same receiver then a new transport session is established whenever a valid subscription is configured and the transport session MUST be closed if the subscription configuration is removed or changed such that the subscription is no longer valid.

  - For YANG Push transports that specify a shared transport session for all subscriptions to the same receiver then a new transport session is established for the first valid configured subscription. 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 14, 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 receiver 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. Any counters associated with the subscription are also re-initialized in a manner that is consistent with a client unconfiguring and then reconfiguring the subscription.

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. Any counters associated with the receiver are also re-initialized in a manner that is consistent with a client unconfiguring and then reconfiguring the receiver configuration.

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-yang-push-2 YANG module Section 13.2, and also via the capabilities module Section 15.2.

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.

* 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 v2 notifications

* optionally includes DSCP settings to use for the transport.

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 13.2.

```
+---x establish-subscription {dynamic}?
|  +---w input
|  |  +---w id?               subscription-id
|  |  +---w description?      string
|  |  +---w target
|  |  |  +---w datastore?       identityref
|  |  |  +---w (filter)
|  |  |     +--:(path)
|  |  |     |  +---w path       ypath
|  |  |     +--:(subtree)
|  |  |     |  +---w subtree    <anydata>
|  |  |     +--:(xpath)
|  |  |        +---w xpath      yang:xpath1.0 {yp2: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 *TODO* (Section 2.4.6)?

9.4.2. Modifying a Dynamic Subscription

The _modify-subscription_ RPC allows a subscriber to request the modification of parameters associated with a dynamic subscription. It uses the same parameters as the _establish-subscription_ RPC defined in Section 9.4.1, except that the _id_ leaf is mandatory.

If the modification to the subscription is accepted by the publisher then it is processed as per Section 9.1.3, otherwise an error is returned to the _modify-subscription_ RPC and the subscription is left unmodified.

The publisher MUST reply to the _modify-subscription_ RPC before sending any subscription lifecycle notifications, i.e., a pair of _subscription-terminated_/_subscription-started_ notifications, or a_subscription-modified_ notification.

Below is a tree diagram for the _modify-subscription_ YANG RPC. All objects contained in this tree are described in the YANG module in Section 13.2.

```
+---x modify-subscription {dynamic}?
|  +---w input
|     +---w id                subscription-id
|     +---w description?      string
|     +---w target
|     |  +---w datastore?       identityref
|     |  +---w (filter)
|     |     +--:(path)
|     |     |  +---w path       ypath
|     |     +--:(subtree)
|     |     |  +---w subtree    <anydata>
|     |     +--:(xpath)
|     |        +---w xpath      yang:xpath1.0 {yp2: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 dscp?             inet:dscp
```

Figure 15: modify-subscription YANG RPC

A publisher MAY reject the "modify-subscription" RPC for various reasons, as described in *TODO*.

9.4.3. 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 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 subscription id immediately after either the RPC operation completes or the _subscription-terminated_ notification (Section 9.2.3) 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 13.2.

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

Figure 16: delete-subscription YANG RPC

9.4.4. 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 id, 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 subscription id immediately after the _subscription-terminated_ notification (Section 9.2.3) 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 13.2.

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

Figure 17: kill-subscription YANG RPC

9.4.5. 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 v2 achieves this in several ways:

* the _complete_ flag in the _update_ notification, or the equivalent _update-complete_ notification, are used to signal when all data for a periodic collection event has been enqueued by the publisher. This allows clients to delete stale information and monitor the performance and behavior of the publisher.

* publishers use buffers when enqueuing traffic on to a transport to a receiver, but if that buffer becomes full then the transport specification indicates whether update messages should be dropped, or the subscription should be terminated. In that case that a dynamic subscription is terminated, the client may attempt to re-created the subscription. For configured subscriptions that have been terminated, the publisher should attempt to periodically recreate the subscription or reconnect the 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].

* 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.

Both configured and dynamic subscriptions are represented in the list _ietf-yang-push-2-config:datastore-telemetry/subscriptions/subscription_. Dynamic subscriptions are only present in the list when they are active, and are removed as soon as they are terminated. Whereas, configured subscriptions are always present in the list when they are configured, regardless of whether they are active.

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

A publisher SHOULD reject a request for a subscription if it is unlikely that the publisher will be able to fulfill the terms of that subscription request.

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.

It is entirely possible that a subscription may be reasonable at the time that it is created, but other changes to configuration or operational data or load in the system means that the subscription can no longer be honored.

TODO - Do we allow servers to reduce the period of the subscription to allow it to be kept active? E.g., with a subscription modification notification? E.g., see issue 41 (https://github.com/rgwilton/draft-yp-observability/issues/41)

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 v2 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-yang-push-2-capabilities YANG module, defined in Section 15.2) that augments ietf-system-capabilities.

The ietf-yang-push-2-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-yang-push-2-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 notc:subscription-capabilities
        |     |  +--ro notc:max-nodes-per-update?                uint32
        |     |  +--ro notc:periodic-notifications-supported?
        |     |  |       notification-support
        |     |  +--ro (notc:update-period)?
        |     |  |  +--:(notc:minimum-update-period)
        |     |  |  |  +--ro notc:minimum-update-period?        uint32
        |     |  |  +--:(notc:supported-update-period)
        |     |  |     +--ro notc:supported-update-period*      uint32
        |     |  +--ro notc:on-change-supported?
        |     |  |       notification-support {yp:on-change}?
        |     |  +--ro notc:minimum-dampening-period?            uint32
        |     |  |       {yp:on-change}?
        |     |  +--ro notc:supported-excluded-change-type*     union
        |     |          {yp:on-change}?
        |     +--ro yp2ca:datastore-telemetry
        |        +--ro yp2ca:periodic-notifications-supported?
        |        |       notification-support
        |        +--ro (yp2ca:update-period)?
        |        |  +--:(yp2ca:minimum-update-period)
        |        |  |  +--ro yp2ca:minimum-update-period?        uint32
        |        |  +--:(yp2ca:supported-update-period)
        |        |     +--ro yp2ca:supported-update-period*      uint32
        |        +--ro yp2ca:on-change-supported?
        |                notification-support
        +--ro notc:subscription-capabilities
        |  +--ro notc:max-nodes-per-update?               uint32
        |  +--ro notc:periodic-notifications-supported?
        |  |       notification-support
        |  +--ro (notc:update-period)?
        |  |  +--:(notc:minimum-update-period)
        |  |  |  +--ro notc:minimum-update-period?        uint32
        |  |  +--:(notc:supported-update-period)
        |  |     +--ro notc:supported-update-period*      uint32
        |  +--ro notc:on-change-supported?
        |  |       notification-support {yp:on-change}?
        |  +--ro notc:minimum-dampening-period?           uint32
        |  |       {yp:on-change}?
```

```
|  +--ro notc:supported-excluded-change-type*     union
|  |       {yp:on-change}?
|  +--ro inotenv:notification-metadata
|     +--ro inotenv:notification-envelope?   boolean
|     +--ro inotenv:metadata
|        +--ro inotenv:hostname-sequence-number?   boolean
+--ro yp2ca:datastore-telemetry
|  +--ro yp2ca:periodic-notifications-supported?
|  |       notification-support
|  +--ro (yp2ca:update-period)?
|  |  +--:(yp2ca:minimum-update-period)
|  |  |  +--ro yp2ca:minimum-update-period?        uint32
|  |  +--:(yp2ca:supported-update-period)
|  |     +--ro yp2ca:supported-update-period*      uint32
|  +--ro yp2ca:on-change-supported?
|  |       notification-support
|  +--ro yp2ca:transport
|     +--ro yp2ca:transport-capability* [transport-protocol]
|        +--ro yp2ca:transport-protocol    identityref
|        +--ro yp2ca:security-protocol?    identityref
|        +--ro yp2ca:encoding-format*      identityref
+--ro yp2ca:distributed-notifications-supported?   boolean
```

Figure 18: 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 Versioning

Many receivers will want to know what the schema is associated with a subscription and whether that schema has changed, e.g., due to a changing in software on the publisher.

Various mechanisms are available to help receivers or collectors learn or monitor the schema associated with a subscription:

1. The device schema is available in the YANG library module ietf-yang-library.yang as defined in [RFC8525]. YANG library also provides a simple "yang-library-change" notification that informs the subscriber that the library has changed, or alternatively, the publisher may support an on-change telemetry subscription to

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. Distributed Notifications - Multiple Publishing Processes

Distributed notifications is the concept of allowing subscriptions to be served from multiple different agent publisher processes on the same device. As a simple example, an implementation could choose to have separate publishing processes for each linecard in a network device, so that data that is local to that linecard can be published directly from the linecard, perhaps directly from the linecard data interfaces, without been sent to a Management Processor card that could act a bottleneck.

The YANG Push 2 specification supports distributed notifications as described in [I-D.ietf-netconf-distributed-notif], and using the terminology defined therewithin, but with some differences due to the different YANG data models, which are described below.

It is OPTIONAL for YANG Push 2 publishers to support multiple publisher agents since they always have the option of handling all subscriptions from a single publisher process on each server. Receivers and consumers of YANG Push 2 subscription data MUST accommodate servers that publish their data from multiple Message Publishing processes, e.g., they may need to correlate the update-complete notification across multiple publishing processes.

Publishers serving a subscription MUST only send the data for a given YANG data node from a single Message Publisher process. Publishers MAY change which Message Publisher process is sending the data for a given YANG data node over time, e.g., to load-balance work.

The differences for supporting [I-D.ietf-netconf-distributed-notif] with YANG Push 2 are:

* the publisher-id leaf, that identities the Message Publisher ID of the publishing process, is augmented in the envelope notification [I-D.draft-ietf-netconf-notif-envelope] rather than the _push-update_ and _push-change-update_ messages. This means that all messages sent by any Message Publishers (e.g., including Subscription Lifecyle Notifications) will indicate which Message Publisher sent the message.

* the publisher-id leaf has a default value of 0. The publisher-id of 0 is reserved for the Publisher Parent process and MUST NOT be allocated to any Publisher Agent processes, since that allows for the publisher-id leaf to be omitted from notification messages sent from the Publisher Parent.

* the _subscription-started_ and _subscription-modified_ notifications have a _message-publishers/publisher-id_ leaf-list that identifies all Message Publishers that will send messages as part of the subscription. This leaf-list defaults to a single entry of 0, so MAY be ommitted if there is only a single Message Publisher and it has an publisher-id of 0.

  - As per [I-D.ietf-netconf-distributed-notif], the list of Message Publishers serving an active subscription can change during the lifetime of the subscription and the Publisher Parent MUST send a _subscription-modified_ notification of any change in publisher-ids.

  - Yang Push 2 does not return the list of publisher-ids as part of the establish-subscription response output because the dynamic subscription will receive a _subscription-started_ notification instead, which also provides a more robust mechanism consistent with configured subscriptions.

* the _subscriptions/subscription_ has a _message-publishers/publisher-id_ leaf-list to report the current list of Message Publishers associated with a subscription.

* A capability flag is used to indicate whether the server might use distributed notifications.

* the _update-complete_ notifications, or the equivalent _complete_ leaf as part of an _update_ notification are generated per Message Publisher process, and are scoped as such. I.e., this may require clients to count and correlate update complete indications across Message Publishers for a given subscription.

13. Core YANG Push v2 YANG Data Model

13.1. ietf-yang-push-2 YANG tree

This section shows the full tree output for ietf-yang-push-2 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-yang-push-2

  rpcs:
    +---x establish-subscription {dynamic}?
    |  +---w input
    |  |  +---w id?                subscription-id
    |  |  +---w description?       string
    |  |  +---w target
    |  |  |  +---w datastore?        identityref
    |  |  |  +---w (filter)
    |  |  |     +--:(path)
    |  |  |     |  +---w path        ypath
    |  |  |     +--:(subtree)
    |  |  |     |  +---w subtree     <anydata>
    |  |  |     +--:(xpath)
    |  |  |        +---w xpath       yang:xpath1.0 {yp2: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 modify-subscription {dynamic}?
    |  +---w input
    |     +---w id                subscription-id
    |     +---w description?      string
    |     +---w target
    |     |  +---w datastore?       identityref
    |     |  +---w (filter)
    |     |     +--:(path)
    |     |     |  +---w path       ypath
    |     |     +--:(subtree)
```

```
        |      |      |  +---w subtree    <anydata>
        |      |      +--:(xpath)
        |      |         +---w xpath      yang:xpath1.0 {yp2: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 dscp?             inet:dscp
        +---x delete-subscription {dynamic}?
        |  +---w input
        |     +---w id    subscription-id
        +---x kill-subscription {dynamic}?
           +---w input
              +---w id    subscription-id

      notifications:
        +---n subscription-started
        |  +--ro id                    subscription-id
        |  +--ro description?          string
        |  +--ro target
        |  |  +--ro datastore?                 identityref
        |  |  +--ro (filter)
        |  |  |  +--:(path)
        |  |  |  |  +--ro path                 ypath
        |  |  |  +--:(subtree)
        |  |  |  |  +--ro subtree              <anydata>
        |  |  |  +--:(xpath)
        |  |  |     +--ro xpath                yang:xpath1.0 {yp2:xpath}?
        |  |  +--ro schema-id?                 string
        |  |  +--ro yang-library-content-id?
        |  |            -> /yanglib:yang-library/content-id
        |  +--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
        |  +--ro message-publishers
        |     +--ro publisher-id*   uint32
        +---n subscription-modified
        |  +--ro id                    subscription-id
        |  +--ro description?          string
        |  +--ro target
        |  |  +--ro datastore?                 identityref
        |  |  +--ro (filter)
        |  |  |  +--:(path)
```

```
      |  |  |  |  +--ro path                ypath
      |  |  |  +--:(subtree)
      |  |  |  |  +--ro subtree             <anydata>
      |  |  |  +--:(xpath)
      |  |  |     +--ro xpath               yang:xpath1.0 {yp2:xpath}?
      |  |  +--ro schema-id?                string
      |  |  +--ro yang-library-content-id?
      |  |           -> /yanglib:yang-library/content-id
      |  +--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
      |  +--ro message-publishers
      |  |  +--ro publisher-id*   uint32
      |  +--ro reason                subscription-change
      +---n subscription-terminated
      |  +--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)?
      |  |     +--:(merge)
      |  |     |  +--ro merge?         <anydata>
      |  |     +--:(replaced-by)
      |  |     |  +--ro replaced-by?   <anydata>
      |  |     +--:(deleted)
      |  |        +--ro deleted?       empty
      |  +--ro complete?           boolean
      +---n update-complete
         +--ro id?   subscription-id
```

Figure 19: YANG tree for YANG Push v2 Module Tree Output

13.2. ietf-yang-push-2 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-yang-push-2.yang#0.1.0"

```
module ietf-yang-push-2 {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push-2";
  prefix yp2;

  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)";
  }
  import ietf-yang-library {
    prefix yanglib;
    reference
      "RFC 8525: YANG Library";
  }
  import ietf-yp-notification {
    prefix iypn;
    reference
     "draft-ietf-netconf-notif-envelope: Extensible YANG Model for
      YANG-Push Notifications

      TODO: RFC Editor please replace with latest draft/RFC version
     at publication time.";
```

```
  }

  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
     version 2, 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 Datastore Telemetry (YANG Push version 2)";
  }

  /*
   * 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.";
    }

    identity conflicting-identifier {
      base subscription-terminated-reason;
      description
        "The subscription identifier conflicts with another
         subscription.

         This can prevent a dynamic subscription from being
         established, or cause a dynamic subscription to be terminated
         if a configured subscription with the same id is created.";

      // TODO - Should there be separate error codes for creating a
      // subscription vs terminating an existing one?
    }


    /* 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-id {
      type string {
         length "1..64";
         pattern '[A-Za-z0-9._-]+';
       }
       description
         "A used friendly string identifier for a subscription.";
```

```
    }

    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 a
```

```
            receiver";
        }
        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";
    }

    typedef subscription-change {
      type enumeration {
        enum new-subscription {
          description
            "This is a new subscription.";
        }
        enum deleted {
          description
            "The subscription has been unconfigured or deleted via
             a delete-subscription RPC";
        }
        enum killed {
          description
            "A dynamic subscription has been killed via a
             kill-subscription RPC";
        }
        enum receiver-added {
          description
            "A new receiver has been added to a configured
             subscription, and has connected successfully.";
        }
        enum receiver-removed {
          description
            "A receiver has been removed from a configured
             subscription, or has become disconnected.";
        }
        enum config-changed {
          description
            "The subscription configuration has been changed,
             requiring the subscription to be restarted.";
        }
```

```
    }

    description
      "Indicates the reason why a subscription has changed.";
  }

  /*
   * 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 publishers {
    description
      "Grouping describes the list of publisher-ids";
    leaf-list publisher-id {
      type uint32;
      default 0;
      config false;
      description
        "Identifies the software process which publishes notification
        messages (e.g., processor 1 on line card 1). This field
        is used to notify the receiver which publisher processes
        are going to publish. The identifiers are locally unique to
        the Network Node.";
    }
  }

  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 "yp2: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 "yp2: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-id {
      description
        "This grouping describes the subscription identifier.";

      leaf id {
        type subscription-id;
        mandatory true;
        description
          "The unique identifier for the subscription.";
      }
    }

    grouping client-subscription-id {
      description
```

```
        "This grouping describes a subscription identifier that
         cannot start with 'dyn-', which is reserved for dynamically
         created subscriptions.";

      leaf id {
        type subscription-id {
          pattern "dyn-.*" {
            modifier "invert-match";
          }
        }
        description
          "A identifier for the subscription, that MUST be unique
          across all configured and dynamic subscriptions.

          MUST NOT start with 'dyn-' which is reserved for
          dynamic subscriptions created by the publisher.";
      }
    }

    grouping subscription-schema {
      description
        "This grouping describes schema information related to a
         subscription.";

      leaf schema-id {
        type string;
        description
          "Indicates the version of the YANG schema used for this
          subscription.  The schema-id MUST change if the schema
          associated with the subscription changes.  The schema-id
          MAY change if the device schema changes, even if the
          change does not affect the subscription.";
      }

      leaf yang-library-content-id {
        type leafref {
          path "/yanglib:yang-library/yanglib:content-id";
        }
        description
          "Contains the YANG library content identifier.";
      }
    }

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

```
      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 client-subscription-id {
          refine id {
            description
              "A optional identifier for a dynamic
               subscription, that must be unique across all
               configured and dynamic subscriptions.

               MUST NOT start with 'dyn-'.

               If not provided, the publisher MUST generate a unique
               identifier for the subscription, with an identifier
               that starts with 'dyn-', which is returned to the
               client in the RPC reply.";
          }
        }
        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.";
        }
      }
    }

    rpc modify-subscription {
      if-feature "dynamic";
      description
        "This RPC allows a subscriber to modify a dynamic
         subscription's parameters.  If successful, the changed
         subscription parameters remain in effect for the duration of
         the subscription, until the subscription is again modified, or
         until the subscription is terminated.  In the case of an error
         or an inability to meet the modified parameters, the
```

```
          subscription is not modified and the original subscription
          parameters remain in effect.";
      input {
        uses subscription-id;
        uses subscription-common;

        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 id {
        type subscription-id;
        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 id {
        type subscription-id;
        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
      */

     grouping subscription-update-common {
       description
         "Data nodes common to both subscription-started and
          subscription-modified notifications.";

       uses subscription-id;
       uses subscription-common {
         augment "target" {
           description
             "Augments fields to identify the target schema.";
           uses subscription-schema;
         }
       }
       container message-publishers {
         description
           "Holds the publisher-ids for all processes currently
            associated with the subscription";
           uses yp2:publishers;
       }
     }

     notification subscription-started {
       description
```

```
        "This notification indicates that a subscription has started
         and notifications will now be sent.";
         uses subscription-update-common;
    }

    notification subscription-modified {
      description
        "This notification indicates that subscription paramaters have
         changed.";
      uses subscription-update-common;

      leaf reason {
        type subscription-change;
        mandatory true;
        description
          "Gives a hint for the message recipient as to why the
           notification has been sent.";
      }
    }

    notification subscription-terminated {
      description
        "This notification indicates that a subscription has been
         terminated.";
      uses subscription-id;
      leaf reason {
        type identityref {
          base subscription-terminated-reason;
        }
        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
          "The identifier of the subscription that is the source of
```

```
          this notification.";
      }

      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.";
```

```
          }
        }
        mandatory true;
        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.";
        }

        choice data {
          description
            "This choice indicates the type of update that is being
             performed at the target path.";
          anydata merge {
            description
              "This contains the updated data that is to be merged with
               the existing data at the target path.  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.

               The snapshot is encoded relative to the combined path
               defined by the common path-prefix and target-path";
```

```
          }
          anydata replaced-by {
            description
              "This contains the updated data that is to replace all
              existing data at the target path.   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.

              The snapshot is encoded relative to the combined path
              defined by the common path-prefix and target-path";
          }
          leaf deleted {
            type empty;
            description
              "This indicates that the data at the target path has been
               deleted.";
          }
        }
      }

      leaf complete {
        type boolean;
        default "false";
        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.";
      }
    }

    notification update-complete {
      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;
        description
          "The name of the subscription that is the source of
           this notification.";
      }
    }

    sx:augment-structure "/iypn:envelope" {
      leaf publisher-id {
        type uint32;
        default 0;
        description
          "Identifies the software process which publishes notification
           messages (e.g., processor 1 on line card 1). This field
           is used to notify the receiver which publisher process
           published which message. The identifier is locally unique to
           the Network Node.";
      }
    }
  }
  <CODE ENDS>
```

Figure 20: YANG module ietf-yang-push-2

14. Configured Subscription YANG Data Model

This document specifies the ietf-yang-push-2-config YANG module Section 14.2 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 13.2. In the operational datastore [RFC8342], the "subscription" list contains entries both for configured and dynamic subscriptions.

```
module: ietf-yang-push-2-config
  +--rw datastore-telemetry!
     +--rw subscriptions
        +--rw subscription* [id]
           +--rw id                                  subscription-id
           +--rw description?                        string
           +--rw target
```

```
|  +--rw datastore?          identityref
|  +--rw (filter)
|     +--:(path)
|     |  +--rw path          ypath
|     +--:(subtree)
|     |  +--rw subtree       <anydata>
|     +--:(xpath)
|     |  +--rw xpath         yang:xpath1.0 {yp2: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 receiver
|       -> /datastore-telemetry/receivers/receiver/name
+--ro status?
|       yp2:subscription-status
+--ro type?
|       yp2:subscription-type
+--ro encoding?                              yp2:encoding
+--ro message-publishers
|  +--ro publisher-id*   uint32
+--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-notifications?
|  |       yang:zero-based-counter64
|  +--ro terminated-notifications?
|  |       yang:zero-based-counter64
|  +--ro update-notifications?
|  |       yang:zero-based-counter64
|  +--ro periodic-collections?
|  |       yang:zero-based-counter64
|  +--ro excluded-events?
|  |       yang:zero-based-counter64
|  +--ro receiver-disconnects?
|          yang:zero-based-counter64
+---x reset
```

```
  augment /yp2:subscription-started/yp2:target:
    +-- filter-ref?   filter-ref
  augment /yp2:subscription-modified/yp2:target:
    +-- filter-ref?   filter-ref
```

Figure 21: 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-yang-push-2-config YANG module.

14.1. ietf-yang-push-2-config YANG tree

This section shows the full tree output for ietf-yang-push-2-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-yang-push-2-config
  +--rw datastore-telemetry!
     +--rw filters
     |  +--rw filter* [name]
     |     +--rw name              string
     |     +--rw (filter)
     |        +--:(path)
     |        |  +--rw path        ypath
     |        +--:(subtree)
     |        |  +--rw subtree     <anydata>
     |        +--:(xpath)
     |           +--rw xpath       yang:xpath1.0 {yp2:xpath}?
     +--rw subscriptions
     |  +--rw subscription* [id]
     |     +--rw id                                  subscription-id
     |     +--rw description?                        string
     |     +--rw target
     |     |  +--rw datastore?          identityref
     |     |  +--rw (filter)
     |     |     +--:(path)
     |     |     |  +--rw path          ypath
     |     |     +--:(subtree)
     |     |     |  +--rw subtree       <anydata>
     |     |     +--:(xpath)
     |     |     |  +--rw xpath         yang:xpath1.0 {yp2: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 receiver
|      |       -> /datastore-telemetry/receivers/receiver/name
|      +--ro status?
|      |       yp2:subscription-status
|      +--ro type?
|      |       yp2:subscription-type
|      +--ro encoding?                              yp2:encoding
|      +--ro message-publishers
|      |  +--ro publisher-id*   uint32
|      +--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-notifications?
|      |  |       yang:zero-based-counter64
|      |  +--ro terminated-notifications?
|      |  |       yang:zero-based-counter64
|      |  +--ro update-notifications?
|      |  |       yang:zero-based-counter64
|      |  +--ro periodic-collections?
|      |  |       yang:zero-based-counter64
|      |  +--ro excluded-events?
|      |  |       yang:zero-based-counter64
|      |  +--ro receiver-disconnects?
|      |          yang:zero-based-counter64
|      +---x reset
|   +--rw receivers
|      +--rw receiver* [name]
|         +--rw name                        string
|         +--rw encoding                    yp2: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 /yp2:subscription-started/yp2:target:
       +-- filter-ref?   filter-ref
     augment /yp2:subscription-modified/yp2:target:
       +-- filter-ref?   filter-ref
```

Figure 22: YANG tree for YANG Push v2 Config Module Tree Output

14.2. ietf-yang-push-2-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-yang-push-2-config.yang#0.1.0"

```
module ietf-yang-push-2-config {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-push-2-config";
  prefix yp2co;

  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-yang-push-2 {
    prefix yp2;
    reference
```

```
        "RFC XXXX: YANG Datastore Telemetry (YANG Push version 2)";
    }

    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 version 2
       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 Datastore Telemetry (YANG Push version 2)";
    }

    /*
     * 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 v2 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 {
              length "1..64";
              pattern '[A-Za-z0-9._-]+';
            }
            description
              "A unique name to identify the selection filters.";
          }
          uses yp2: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 "id";
          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 yp2:client-subscription-id;
          uses yp2:subscription-common;

          leaf receiver {
```

```
          type leafref {
            path "/datastore-telemetry/receivers/receiver/name";
          }
          mandatory true;
          description
            "Identifies the 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.";
        // }
```

```
          leaf status {
            type yp2:subscription-status;
            config false;
            description
              "The presence of this leaf indicates that the
               subscription originated from configuration, not through
               a control channel or RPC.  The value indicates the state
               of the subscription as established by the publisher.";
          }

          leaf type {
            type yp2:subscription-type;
            default "configured";
            config false;
            description
              "Whether the subscription is configured or dynamic.";
          }

          leaf encoding {
            type yp2:encoding;
            config false;
            description
              "The type of encoding for notification messages.";
          }

          container message-publishers {
            config false;
            description
              "Holds the publisher-ids for all processes
               currently associated with the subscription";
              uses yp2:publishers;
          }

          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-notifications {
          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-notifications {
        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-notifications {
        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-collections {
        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-events {
        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.";
          }

          leaf receiver-disconnects {
            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";
          }
        }

        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 {
            length "1..64";
            pattern '[A-Za-z0-9._-]+';
          }
          description
            "An arbitrary but unique name for this receiver
            instance.";
        }

        leaf encoding {
  /*        when 'not(../transport) or derived-from(../transport,
```

```
            "yp2:configurable-encoding")';*/
            type yp2: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 yp2:dscp;

          action reset {
            description
              "Resets all configured subscriptions that reference
               this receiver.

               This action is similar to invoking the 'reset' action
               on all subscriptions that reference 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 "/yp2:subscription-started/yp2:target" {

      description
        "Allow a subscription-started notification to include a
         referenced named filter";
      uses filter-ref {
        refine "filter-ref" {
          mandatory false;
        }
      }
    }

    augment "/yp2:subscription-modified/yp2:target" {
```

```
      description
        "Allow a subscription-modified 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 23: YANG module ietf-yang-push-2-config

15. Capabilities YANG Data Model

15.1. ietf-yang-push-2-capabilities YANG tree

This section shows the tree output for ietf-yang-push-2-capabilities YANG module, which augments the ietf-system-capabilities YANG module [RFC9196].

```
module: ietf-yang-push-2-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
    +--ro distributed-notifications-supported?   boolean
  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 v2 Capabilities Module Tree Output

15.2. ietf-yang-push-2-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-yang-push-2-capabilities.yang#0.1.0"
module ietf-yang-push-2-capabilities {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-push-2-capabilities";
  prefix yp2ca;

  import ietf-system-capabilities {
    prefix sysc;
    reference
      "RFC 9196: YANG Modules Describing Capabilities for Systems
```

```
        and Datastore Update Notifications";
    }
    import ietf-yang-push-2 {
      prefix yp2;
      reference
        "RFC XXX: YANG Datastore Telemetry (YANG Push version 2)";
    }

    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 Datastore Telemetry (YANG Push version 2)";
    }

    /*
     * 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 yp2-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 yp2-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 yp2: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 yp2:encoding;
            }
            description
              "Indicates supported encoding formats.";
          }
        }
      }
    }

    // YANG Push Lite Capabilities
    augment "/sysc:system-capabilities" {
      description
        "Adds system level capabilities for YANG Push Lite";
      uses yp2-capabilities;

      leaf distributed-notifications-supported {
        type boolean;
        description
          "Indicates whether the server supports distributed
            notifications and may source message from different
            Message Publishers.";
      }
    }

    augment "/sysc:system-capabilities/yp2ca:datastore-telemetry" {
      description
        "Adds system level Yang Push Lite transport capabilities";
```

```
      uses yp2-transport-capabilities;
    }

    augment "/sysc:system-capabilities/sysc:datastore-capabilities"
          + "/sysc:per-node-capabilities" {
      description
        "Add datastore and node-level capabilities";
      uses yp2-capabilities;
    }
  }
  <CODE ENDS>
```

Figure 25: YANG module ietf-yang-push-2-capabilities

16. 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.

16.1. Use of YANG Push v2 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.

16.2. 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.

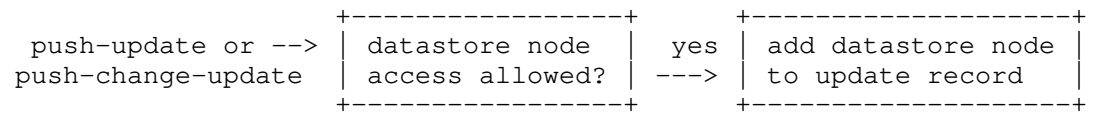


Figure 26: Access Control for Push Updates

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.

16.3. 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-yang-push-2" 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.

17. IANA Considerations

17.1. Namespace URI registrations

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

URI
urn:ietf:params:xml:ns:yang:ietf-yang-push-2
urn:ietf:params:xml:ns:yang:ietf-yang-push-2-config
urn:ietf:params:xml:ns:yang:ietf-yang-push-2-capabilities

Table 1: Namespace URI registrations

For all registrations:

* Registrant Contact: The IESG.

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

18. YANG Module Name registrations

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

Name	Namespace	Prefix
ietf-yang-push-2	urn:ietf:params:xml:ns:yang:ietf-yang-push-2	ypl
ietf-yang-push-2-config	urn:ietf:params:xml:ns:yang:ietf-yang-push-2-config	yplco
ietf-yang-push-2-capabilities	urn:ietf:params:xml:ns:yang:ietf-yang-push-2-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.

In addition, as described in the introduction, the authors would like to acknowledge that this work builds on work of others, such as, Subscribed Notifications, YANG Push, and various extensions to that work.

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-05, 18 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-notif-envelope-05>.

[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., BjÃ¶rklund, 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>.

[RFC9911] SchÃ¶nwÃ¤lder, J., Ed., "Common YANG Data Types", RFC 9911, DOI 10.17487/RFC9911, December 2025, <https://www.rfc-editor.org/rfc/rfc9911>.

Informative References

[Consistency] Wikipedia, "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-19, 13 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-distributed-notif-19>.

[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-25, 28 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-udp-notif-25>.

[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-19, 13 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-distributed-notif-19>.

[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-07, 18 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-network-anomaly-architecture-07>.

[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-12, 13 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-nmop-yang-message-broker-integration-12>.

[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>.

[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 v2 and YANG Push

This non-normative section highlights the significant functional changes where the YANG Push v2 implementation differs from YANG Push. However, the main body of this document, from Section 4 onwards, provides the normative definition of the YANG Push v2 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 v2.

* 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, for both configured and dynamic subscriptions. 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.

* Support for multiple receivers for a configured subscription.

A.2. Changed Functionality

This section documents behavior that exists in both YANG Push and YANG Push v2, but the behavior differs between the two:

* All YANG Push v2 notifications messages use [I-D.draft-ietf-netconf-notif-envelope] rather than [RFC5277] used by YANG Push [RFC8641].

* There is a lot more alignment in data model, behavior, and state machined in YANG Push v2, aiming to minimize differences.

* 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).

  - 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.

  - YANG Push v2 only supports a single receiver per subscription.

* 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.

* Changes related to the configuration model:

  - Subscriptions are identified by a string identifier rather than a numeric identifier. The prefix 'dyn-' is reserved for publisher allocated dyanmic subscription ids. Clients may provide the id to be used for dynamic subscriptions. There is changed handling for conflicting subscription-ids between configured and dynamic.

  - Purpose has been renamed to Description (since it is a more generic term), limited to 1k characters, but also available for dynamic subscriptions.

* On-change dampening:

  - Client configurable on-change dampening has been removed.

  - However, YANG Push v2 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 v2.

* 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 [RFC8639]/[RFC8641]:

  - 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 v2 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.

* An _update-complete_ notification has been defined to inform collectors when a subscription's periodic collection cycle is complete.

* Support for {I-D.ietf-netconf-distributed-notif}} has been added to the base YANG Push v2 specification, as described in Section 12.

* TODO - More operational data on the subscription load and performance.

* All YANG Push v2 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	no-such-subscription-resync
unchanging-selection	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-yang-push-2:update": {
    "id": "interfaces",
    "path-prefix": "/ietf-interfaces:interfaces/interface",
    "snapshot-type": "periodic",
    "observation-time": "2024-10-10T08:00:05.11Z",
    "updates": [
      {
        "target-path": "",
        "merge": {
          "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": false
  }
}
```

Figure 27: 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-yang-push-2:update": {
    "id": "interfaces",
    "path-prefix": "/ietf-interfaces:interfaces/interface",
    "snapshot-type": "periodic",
    "observation-time": "2024-10-10T08:00:05.21Z",
    "updates": [
      {
        "target-path": "",
        "merge": {
          "ietf-interfaces:interface": [
            {
              "name": "eth0",
              "statistics": {
                "in-octets": 123456,
                "out-octets": 654321
              }
            },
            {
              "name": "eth1",
              "statistics": {
                "in-octets": 789012,
                "out-octets": 210987
              }
            }
          ]
        }
      }
    ]
  }
}
```


Figure 28: 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 29: Example YANG Push v2 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-yang-push-2:update": {
    "id": "interfaces",
    "snapshot-type": "on-change-delete",
    "observation-time": "2025-10-10T08:16:15.11Z",
    "updates": [
      {
        "target-path": "/ietf-interfaces:interfaces/interface[name='eth0']",
        "deleted": [null]
      }
    ]
  }
}
```

Figure 30: Example YANG Push v2 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-yang-push-2:update": {
    "id": "interfaces",
    "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 31: Example YANG Push v2 on-change delete message

C.4. NETCONF Dynamic Subscription RPC examples

The examples in this section illustrate NETCONF RPCs for establishing and deleting dynamic subscriptions using YANG Push v2. The examples include one successfully establishing a subscription, and a second to illustrate how errors are returned if a request to establish the subscription fails. Examples of the _update_ and subscription lifecycle notifications have been given in the previous section.

C.4.1. Successful periodic subscription

The subscriber sends an "establish-subscription" RPC with the parameters listed in Section 9.4.1 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-yp-lite">
    <name>if-stats-sub</name>
    <description>Subscribe to interface statistics</description>
    <target>
      <datastore xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
        ds:operational
      </datastore>
      <path>/ietf-interfaces:interfaces/interface/statistics</path>
    </target>
    <update-trigger>
      <periodic>
        <period>3000</period>
      </periodic>
    </update-trigger>
    <encoding>json</encoding>
  </establish-subscription>
</netconf:rpc>
```

Figure 32: Example establish-subscription RPC for interface statistics

If a publisher is happy to accept the subscription, then it returns a positive response that includes the "id" of the accepted subscription. For example,

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

Figure 33: Example establish-subscription RPC successful reply

Once established, the publisher would send a _subscription-started_ notification message followed by _update_ notification messages at the requested periodic cadence.

C.4.2. Failed periodic subscription

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.4.3. "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.5. Distributed Notifications Subscription

This simple example illustrates how messages may look like when using distributed notifications, as described in Section 12. This example is for a very simple periodic subscription to the ietf-interfaces YANG data model.

The first message is a subscription started message that indicates that there are 2 additional _Publisher Agents_ with publisher-ids of 4 and 7, e.g., perhaps representing a device with two linecards inserted into slots 4 and 7, each with their owner separate Message Publisher process. This message has a publisher-id of 0 because it is sent by the Publisher Parent, which, because this field has the default value, could have been elided from the message.

```
{
  "ietf-yp-notification:envelope": {
    "event-time": "2025-10-10T08:00:05.22Z",
    "hostname": "example-router",
    "sequence-number": 23,
    "ietf-yang-push-2:publisher-id": 0
    "contents": {
      "ietf-yang-push-2:subscription-started": {
        "id": "interfaces",
        "target": {
          "datastore": "ietf-datastores:operational",
          "path": "/ietf-interfaces:interfaces"
        },
        "update-trigger": {
          "periodic": {
            "period": 3000,
            "anchor-time": "2025-01-01T00:00:00.00Z"
          }
        },
        "message-publishers": {
          "publisher-id" = [0, 4, 7]
        }
      }
    }
  }
}
```

Figure 34: Example distributed notification subscription started message

The second message is an example of a periodic update message sent from one of the Publisher Agents on the linecard. In this case the message is sent from the publisher agent on linecard 4, and it has set the _complete_ leaf to indicate that the message represents all data for that periodic collection from that publisher.

```
{
  "ietf-yp-notification:envelope": {
    "event-time": "2025-10-10T08:00:05.22Z",
    "hostname": "example-router",
    "sequence-number": 18,
    "ietf-yang-push-2:publisher-id": 4
    "contents": {
      "ietf-yang-push-2:update": {
        "id": 1011,
        "snapshot-type": "periodic",
        "observation-time": "2025-10-10T08:00:05.11Z",
        "updates": [
          {
            "target-path": "ietf-interfaces:interfaces/interface",
            "merge": {
              "ietf-interfaces:interfaces": {
                "ietf-interfaces:interface": [
                  {
                    "name": "eth4/0",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": true,
                    "ietf-interfaces:oper-status": "up",
                    "ietf-interfaces:admin-status": "up"
                  },
                  {
                    "name": "eth4/1",
                    "type": "iana-if-type:ethernetCsmacd",
                    "enabled": true,
                    "ietf-interfaces:oper-status": "down",
                    "ietf-interfaces:admin-status": "up"
                  }
                ]
              }
            }
          }
        ],
        complete = true
      }
    }
  }
}
```

Figure 35: Example distributed notification update message

Not shown here, but because there were 3 publisher-ids listed in the subscription-started message means that each Message Publisher must send at least one message for each periodic subscription returning any data associated with the subscription from that Message Publisher

and also indicating that the periodic collection is complete by either setting complete = true or sending an _update-complete_ notification.

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. Issue 27 (https://github.com/rgwilton/draft-yp-observability/issues/27)

2. Do we need to fold in any text from RFC 8640? and RESTCONF? Issue 26 (https://github.com/rgwilton/draft-yp-observability/issues/26)

D.2. Issue related to Terminology/Definitions

1. Should we use the object terminology? Tracked as editorial, in Issue 15 (https://github.com/rgwilton/draft-yp-observability/issues/15)

D.3. Issues related to YANG Push v2 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. Issue 29 (https://github.com/rgwilton/draft-yp-observability/issues/29)

2. Advertising subscription schema: Issue 11 (https://github.com/rgwilton/draft-yp-observability/issues/11)

D.5. Issues related to Datastore Event Streams & message format

1. Agree format and usage of _update_ notification. Issue 32 (https://github.com/rgwilton/draft-yp-observability/issues/32)

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

D.6.1. Issues related to Transports:

1. James: We also need to add some text into security section.
    * Rob: Is this transport specific, or related to application layer authorization?
2. 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:

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. Note some of these apply or impact dynamic subscriptions as well. Issue 30 (https://github.com/rgwilton/draft-yp-observability/issues/30)

D.7.2. Issues related to dynamic subscriptions:

1. 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), is it possible to unify this so we don’t need large extra separate documents.

D.8. Issues related to Subscription Lifecycle

1. 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? Issue 11 (https://github.com/rgwilton/draft-yp-observability/issues/11)

2. 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? Issue 20 (https://github.com/rgwilton/draft-yp-observability/issues/20)

3. 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? Issue 22 (https://github.com/rgwilton/draft-yp-observability/issues/22)

4. Should we have a YANG Action to reset a receiver or a subscription? E.g., discussed in Section 9.3.4. Issue 24 (https://github.com/rgwilton/draft-yp-observability/issues/24)

5. Should we support configurable subscription-level keepalives? Issue 14 (https://github.com/rgwilton/draft-yp-observability/issues/14)

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.

   * Tracked by Issue 31 (https://github.com/rgwilton/draft-yp-observability/issues/31)

2. Should dynamic subscriptions use the same receivers structure as for configured subscriptions, or should they be inline in the configured subscription? Also tracked by Issue 31 (https://github.com/rgwilton/draft-yp-observability/issues/31)

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)?

2. For on-change, should a subscription be rejected (or not brought up) if there are no on-change notifiable nodes?

3. 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.

All tracked via Issue 18 (https://github.com/rgwilton/draft-yp-observability/issues/18)

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 v2, 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.

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.

All tracked via Issue 33 (https://github.com/rgwilton/draft-yp-observability/issues/33)

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 many 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.

7. If subscription parameters change then force subscription down and up again, issue 14 (https://github.com/rgwilton/draft-yp-observability/issues/14)

8. No RPC to modify a dynamic subscription, use delete-subscription then create-subscription.

9. Lifecycle messages are sent per subscription rather than per receiver, and we only support a single receiver per subscription.

10. Encoding is set per subscription, and we don't allow different per-receiver encodings for a subscription with more than one receiver. issue 17 (https://github.com/rgwilton/draft-yp-observability/issues/17)

11. We have a updated-completed flag/notification to allow deleted data to be implicitly detected. Something similar may be added to gNMI. issue 12 (https://github.com/rgwilton/draft-yp-observability/issues/12)

12. We use a string identifier to uniquely identify a subscription rather than a numeric id. Reserve 'dyn-' for server allocated dynamic subscription ids. Agreed config/dynamic conflict policy. Restricted names for fitler-id and receiver name. Don't use a union type (could be added in future). (Dec 8th)

13. Draft renamed from Yang Push Lite to Yang Push 2. (Dec 8th)

14. ietf-yp2-config no longer augments dynamic subscriptions with a reference to a configured filter issue 19 (https://github.com/rgwilton/draft-yp-observability/issues/19)

15. Publisher MUST NOT send more notifications after a subscription terminated message Issue 13 (https://github.com/rgwilton/draft-yp-observability/issues/13).

16. on-change notifications SHOULD send a minimal update but MAY send additional unchanged fields.

Authors' Addresses

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

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

Benoit Claise
Everything OPS
Email: benoit@everything-ops.net

Ebben Aries
Juniper
Email: exa@juniper.net

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

Thomas Graf
Swisscom

Email: Thomas.Graf@swisscom.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2027

P. Andersson
Ionio Systems
M. T. Valaphil
Equinix

7 July 2026

Error List and Error Identities Registries for YANG-driven protocols
draft-pamtv-netconf-error-registries-01

Abstract

This document defines IANA registries for the YANG Protocol Error List and YANG Protocol Error Identities.

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 8 January 2027.

Copyright Notice

Table of Contents

1. Introduction

In order to have a canonical source for all errors and error identities for YANG-driven procotols such as NETCONF [RFC6241] and RESTCONF [RFC8040], two IANA registries are defined: YANG Protocol Error List and YANG Protocol Error Identities. These registries can be extended and updated as needed.

It is an advantage for protocol evolution to have canonical sources to extend, instead of a set of documents that need to be compiled and referenced.

2. Conventions and Definitions

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

3. IANA Considerations

3.1. YANG Protocol Error List Registry

This document defines a registry for the YANG Protocol Error List defined in Appendix A of [RFC6241]. The name of the registry is "YANG Protocol Error List". The review policy for this registry is "IETF Review" [RFC5226]. The registry shall record the following for each entry:

* error-tag, TBD

* error-type, TBD

* error-severity, TBD

* error-info, auxilary information described as data nodes (XML tags).

* Description

* The reference for the document registering the value.

The initial contents of the registry is defined in [RFC6241] and inlined below:

```
error-tag:      in-use
error-type:     protocol, application
error-severity: error
error-info:     none
Description:    The request requires a resource that already is in
                use.

error-tag:      invalid-value
error-type:     protocol, application
error-severity: error
error-info:     none
Description:    The request specifies an unacceptable value for one
                or more parameters.

error-tag:      too-big
error-type:     transport, rpc, protocol, application
error-severity: error
error-info:     none
Description:    The request or response (that would be generated) is
                too large for the implementation to handle.

error-tag:      missing-attribute
error-type:     rpc, protocol, application
error-severity: error
error-info:     <bad-attribute> : name of the missing attribute
                <bad-element> : name of the element that is supposed
                  to contain the missing attribute
Description:    An expected attribute is missing.

error-tag:      bad-attribute
error-type:     rpc, protocol, application
error-severity: error
error-info:     <bad-attribute> : name of the attribute w/ bad value
                <bad-element> : name of the element that contains
                  the attribute with the bad value
Description:    An attribute value is not correct; e.g., wrong type,
                out of range, pattern mismatch.
```

3.2. YANG Protocol Error Identities Registry

This document defines a registry for YANG Protocol Error Identities. The name of the registry is "YANG Protocol Error Identities". The review policy for this registry is "IETF Review" [RFC5226]. The registry shall record the following for each entry:

* Identity name.

* Base identity, if applicable.

* Additional information.

* The reference for the document registering the value.

The initial contents of the registry is defined in [RFC8639] and [RFC8641] and inlined below:

dscp-unavailable
encoding-unsupported
filter-unsupported
insufficient-resources
replay-unsupported

filter-unsupported
insufficient-resources
no-such-subscription

no-such-subscription

datastore-not-subscribable
unchanging-selection

period-unsupported
update-too-big
synchronization-size

4. Normative References

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/rfc/rfc5226>.

[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>.

[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>.

[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>.

[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>.

[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>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Per Andersson
Ionio Systems
Email: per.ietf@ionio.se

Mithun Thai Valaphil
Equinix
Email: mithun.ietf@gmail.com