Network Working Group J. Dai
INTERNET-DRAFT X. Wang
Intended Status: Proposed Standard Y. Kou
Expires: December 18, 2022 L. Zhou
China Information Communication Technologies Group
June 18, 2022

Using NETCONF over QUIC connection
draft-dai-netconf-quic-netconf-over-quic-02

Abstract

The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices. At present, almost all implementations of NETCONF are based on TCP based protocol. QUIC, a new UDP-based, secure and multiplexed transport protocol, can facilitate to improve the transportation performance, information security and resource utility when being used as an infrastructure layer of NETCONF. This document describes how to use the QUIC protocol as the transport protocol of NETCONF(It is so called NETCONFoQUIC).

Status of this Memo

Copyright and License 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 layer, operation layer, message layer and security transport layer.

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, the requirements include the following aspects:

(1). NETCONF is connection-oriented, requiring a persistent connection between peers. This connection MUST provide reliable, 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.

So, 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. At the present, there are a few secure transport protocols that can be used to carry NETCONF:

(1). [RFC6242] specifies how to use secure shell(SSH) as the secure transport layer of NETCONF.

(2). [RFC5539] specifies how to use transport layer security(TLS) as the secure transport layer of NETCONF.

(3). [RFC4743] specifies how to use simple object access protocol(SOAP)as the secure transport layer of NETCONF.

(4). [RFC4744] specifies how to use blocks extensible exchange protocol(BEEP) as the secure transport layer of NETCONF.

However, because of the connection-oriented feature, almost all of the current secure transport protocols used by NETCONF is TCP based. As is well known, TCP has some shortcomings such as head-of-line blocking.

[I-D.ietf-quic-transport] specifies a new transport protocol that has the following features:

(1). UDP based

(2). Stream multiplexing

(3). Stream and connection-level flow control

(4). Low-latency connection establishment

(5). Authenticated and encrypted header and payload

It can be learned from the afore-mentioned information that QUIC is also a proper candidate transport protocol for the secure transport layer of NETCONF. In addition, QUIC can perfectly fix the shortcoming such as head of line blocking of TCP. This document specifies how to use QUIC as the secure transport protocol for QUIC.

In this document, the terms "client" and "server" are used to refer to the two ends of the QUIC connection. The client actively initiates the QUIC connection. The terms "manager" and "agent" are used to refer to the two ends of the NETCONF protocol session. The manager issues NETCONF remote procedure call (RPC) commands, and the agent replies to those commands. Generally, a "manager" is a "client" meanwhile an "agent" is a "server".

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Connection management

3.1. Draft Version Identification

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

NETCONFoQUIC uses the token "NoQ" to identify itself in ALPN and Alt-Svc. Only implementations of the final, published RFC can identify themselves as "NoQ". Until such an RFC exists, implementations MUST NOT identify themselves using this string.

Implementations of draft versions of the protocol MUST add the string "-" and the corresponding draft number to the identifier.

3.2. Connection setup

3.2.1. Version negotiation

QUIC versions are identified using a 32-bit unsigned number, and the version 0x00000000 is reserved to represent version negotiation.

Version negotiation ensures that client and server agree to a QUIC version that is mutually supported.

A server sends a Version Negotiation packet where multiple QUIC versions are listed to the client, the order of the values reflects the server's preference (with the first value being the most preferred version). Reserved versions MAY be listed, but unreserved versions which are not supported by the alternative SHOULD NOT be present in the list.

When received the Version Negotiation packet, Clients MUST ignore any included versions which they do not support.

If both of the server and the client support the QUIC version that uses TLS version 1.3 or greater as its handshake protocol, the afore-mentioned QUIC version should be the preferred QUIC version of the server and the client.

3.2.2. Connection establishment

QUIC connections are established as described in [I-D.ietf-quic-transport]. During connection establishment, NETCONFoQUIC support is indicated by selecting the ALPN token "NoQ" in the crypto handshake.

The peer acting as the NETCONF manager MUST also act as the client meanwhile the peen acting as the NETCONF agent must also act as the server.

The manager should the initiator of the QUIC connection to the agent meanwhile the agent act as a connection acceptor.

3.3. Connection Closure

3.3.1. QUIC connection termination mode

There are following methods to terminate a QUIC connection:

(1) idle timeout: If the idle timeout is enabled, a connection is silently closed and the state is discarded when it remains idle for longer than both the advertised idle timeout and three times the current Probe Timeout (PTO).

(2) immediate close: An endpoint sends a CONNECTION_CLOSE frame (Section 19.19 of [I-D.ietf-quic-transport]) to terminate the connection immediately.

(3) stateless reset: A stateless reset is provided as an option of last resort for an endpoint that does not have access to the state of a connection.

3.3.2. NETCONFoQUIC consideration for connection termination

When a NETCONF session is implemented based on a QUIC connection, the idle timeout should be disabled in order to keep the QUIC connection persistent even if the NETCONF session is idle.

When a NETCONF server receives a <close-session> request, it will gracefully close the NETCONF session. The server must 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 detects any QUIC connection interrupt status, 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 manager or the agent.

4 Stream mapping and usage

[RFC6241] specifies protocol layers of NETCONF, the protocol layers structure can also be seen from figure 1 of this document, it is noted that this figure is just a citation from [RFC6241].

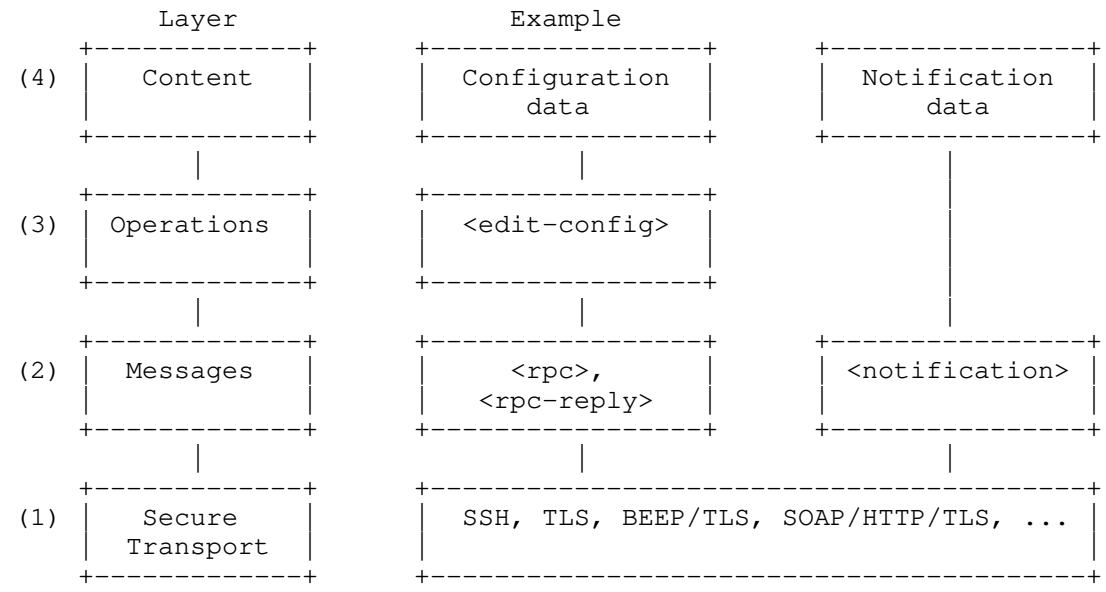

Figure 1: NETCONF Protocol Layers

It can be learned from figure 1 that there are two kinds of main data flow exchanged between manager and agent:

(1) Configuration data from manager to agent.

(2) Notification data from agent to manager.

The two kinds of data flow should 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. Table 1 describes the four types of streams and this table can also be seen from [I-D.ietf-quic-transport].

Bits	Stream Type
0x0	Client-Initiated, Bidirectional
0x1	Server-Initiated, Bidirectional
0x2	Client-Initiated, Unidirectional
0x3	Server-Initiated, Unidirectional

Table 1: Stream ID Types

4.1. Bidirectional stream between manager and agent

The NETCONF protocol uses an RPC-based communication model. So, the configuration data from manager to agent is exchanged based on '<RPC>' (the manager initiating) and '<RPC-Reply>' (sent by the agent) and so on. So the messages used to exchange configuration data should be mapped into one or more bidirectional stream whose stream type is 0.

4.2. Unidirectional stream from agent to manager

There are some notification data exchanged between the agent and the manager. Notification is a server-initiated message indicating that a certain event has been recognized by the server.

Notification messages are initiated by the agent and no reply is needed from the manager. So the messages used to exchange configuration data should be mapped into one unidirectional stream whose stream type is 3.

5 Endpoint Authentication

5.1 using QUIC handshake authentication

NETCONFoQUIC is recommended to use the QUIC version uses TLS version 1.3 or greater. Then, the TLS handshake process can be used for endpoint authentication.

5.1.1. Server Identity

During the TLS negotiation, the client MUST carefully examine the certificate presented by the server to determine if it meets the client's expectations. Particularly, the client MUST check its

understanding of the server hostname against the server's identity as presented in the server Certificate message, in order to prevent man- in-the-middle attacks.

Matching is performed according to the rules below (following the example of [RFC4642]):

o The client MUST use the server hostname it used to open the connection (or the hostname specified in the TLS "server_name" extension [RFC5246]) as the value to compare against the server name as expressed in the server certificate. The client MUST NOT use any form of the server hostname derived from an insecure remote source.

o If a subjectAltName extension of type dNSName is present in the certificate, it MUST be used as the source of the server's identity.

o Matching is case-insensitive.

o A "*" wildcard character MAY be used as the leftmost name component in the certificate. For example, *.example.com would match a.example.com, foo.example.com, etc., but would not match example.com.

o If the certificate contains multiple names then a match with any one of the fields is considered acceptable.

If the match fails, the client MUST either ask for explicit user confirmation or terminate the connection and indicate the server's identity is suspect.

Additionally, clients MUST verify the binding between the identity of the servers to which they connect and the public keys presented by those servers. Clients SHOULD implement the algorithm in Section 6 of [RFC5280] for general certificate validation, but MAY supplement that algorithm with other validation methods that achieve equivalent levels of verification (such as comparing the server certificate against a local store of already-verified certificates and identity bindings).

If the client has external information as to the expected identity of the server, the hostname check MAY be omitted.

5.1.2. Client Identity

The server MUST verify the identity of the client with certificate- based authentication according to local policy to

ensure that the incoming client request is legitimate before any configuration or state data is sent to or received from the client.

5.2 using third-party authentication

A third-party authentication mechanism can also be used for NETCONFoQUIC endpoint authentication. for example, a SASL profile based authentication method can be used.

6. Security Considerations

The security considerations described throughout [RFC5246] and [RFC4741] apply here as well.

This document in its current version does not support third-party authentication (e.g., backend Authentication, Authorization, and Accounting (AAA) servers) due to the fact that TLS does not specify this way of authentication and that NETCONF depends on the transport protocol for the authentication service. If third-party authentication is needed, BEEP or SSH transport can be used.

An attacker might be able to inject arbitrary NETCONF messages via some application that does not carefully check exchanged messages or deliberately insert the delimiter sequence in a NETCONF message to create a DoS attack. Hence, applications and NETCONF APIs MUST ensure that the delimiter sequence defined in Section 2.1 never appears in NETCONF messages; otherwise, those messages can be dropped, garbled, or misinterpreted. If the delimiter sequence is found in a NETCONF message by the sender side, a robust implementation of this document should warn the user that illegal characters have been discovered. If the delimiter sequence is found in a NETCONF message by the receiver side (including any XML attribute values, XML comments, or processing instructions), a robust implementation of this document must silently discard the message without further processing and then stop the NETCONF session.

Finally, this document does not introduce any new security considerations compared to [RFC4742].

7 IANA Considerations

This document creates a new registration for the identification of NETCONFoQUIC in the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" registry established in [RFC7301].

The "noq" string identifies NETCONFoQUIC:

Protocol: NETCONFoQUIC

Identification Sequence: 0x6e 0x6f 0x71 ("noq")

Specification: This document

In addition, it is requested for IANA to reserve a UDP port TBD for 'NETCONF over QUIC'.

8 Acknowledgements

This document is written by referring [I-D.ietf-quic-transport] edited by Jana Iyengar and Martin Thomson and [I-D.ietf-quic-http] edited by Mike Bishop, and from [RFC7858] authored by Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul Hoffman.

Many thanks to all the afore-mentioned editors and authors.

9 References

9.1 Normative References

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-27 (work in progress), Fegruary 2019.

[I-D.ietf-quic-tls] M. Thomson. and S. Turner., " Using TLS to Secure QUIC", draft-ietf-quic-tls-27 (work in progress), February 2019.

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6241] Enns, R., "NETCONF Configuration Protocol", RFC 6241, December 2011.

[RFC6242] M. Wasserman., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC5539] Dierks, T. and E. Rescorla, "NETCONF over Transport Layer Security (TLS)", RFC 5539, May 2009.

[RFC4743] T. Garddard, "Using NETCONF over the Simple Object Access Protocol (SOAP)", RFC 4743, December 2006.

[RFC4744] E. Lear, "Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP)", RFC 4743, December 2006.

[I-D.ietf-quic-http] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-18 (work in progress), January 2019.

9.2 Informative References

[RFC5246] M. Badra, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6101] M. Rose, "The Secure Sockets Layer (SSL) Protocol Version 3.0", RFC 6101, August 2011.

[RFC3080] M. Rose, "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

Authors' Addresses

Jinyou Dai
China Information Communication Technologies Group.
Gaoxin 4th Road 6#
Wuhan, Hubei 430079
China

Email: djy@fiberhome.com

Xueshun Wang
China Information Communication Technologies Group.
Gaoxin 4th Road 6#
Wuhan, Hubei 430079
China

Email: xswang@fiberhome.com

Yang Kou
China Information Communication Technologies Group.
Gaoxin 4th Road 6#
Wuhan, Hubei 430079
China

Email: ykou@fiberhome.com

Lifen Zhou
China Information Communication Technologies Group.
Gaoxin 4th Road 6#
Wuhan, Hubei 430079
China

Email: lfzhou@fiberhome.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 9 January 2023

T. Zhou
G. Zheng
Huawei
E. Voit
Cisco Systems
T. Graf
Swisscom
P. Francois
INSA-Lyon

8 July 2022

Subscription to Distributed Notifications
draft-ietf-netconf-distributed-notif-04

Abstract

This document describes extensions to the YANG notifications subscription to allow metrics being published directly from processors on line cards to target receivers, while subscription is still maintained at the route processor in a distributed forwarding system.

Requirements Language

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].

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 9 January 2023.

Table of Contents

1. Introduction

The mechanism to support a subscription of a continuous and customized stream of updates from a YANG datastore is defined in [RFC8639] and [RFC8641]. Requirements for Subscription to YANG Datastores are defined in [RFC7923]

By streaming data from publishers to receivers, much better performance and fine-grained sampling can be achieved than with polling. In a distributed forwarding system, the packet forwarding

is delegated to multiple processors on line cards. To not to overwhelm the route processor resources, it is not uncommon that data records are published directly from processors on line cards to target Receivers to further increase efficiency on the routing system.

This document complement the general subscription requirements defined in section 4.2.1 of [RFC7923] by the paragraph: A Subscription Service MAY support the ability to export from multiple software processes on a single routing system and expose the information which software process produced which message to maintain data integrity.

2. Terminologies

The following terms are defined in [RFC8639] and are not redefined here:

Subscriber

Publisher

Receiver

Subscription

In addition, this document defines the following terms:

Global Subscription: is the Subscription requested by the subscriber. It may be decomposed into multiple Component Subscriptions.

Component Subscription: is the Subscription that defines a data source which is managed and controlled by a single Publisher.

Global Capability: is the overall subscription capability that the group of Publishers can expose to the Subscriber.

Component Capability: is the subscription capability that each Publisher can expose to the Subscriber.

Master: is the Publisher that interacts with the Subscriber to deal with the Global Subscription. It decomposes the Global Subscription to multiple Component Subscriptions and interacts with the Agents.

Agent: is the Publisher that interacts with the Master to deal with the Component Subscription and pushing the data to the Receiver.

Observation Domain: An Observation Domain is the largest set of Observation Points for which metrics can be collected by a metering process. For example, a router line card may be an Observation Domain if it is composed of several interfaces, each of which is an Observation Point. In the YANG notification messages it generates, the Observation Domain includes its Observation Domain ID, which is unique per publisher process. That way, the collecting process can identify the specific Observation Domain from the publisher that sends the YANG notification messages. Every Observation Point is associated with an Observation Domain.

Observation Domain ID: A 32-bit identifier of the Observation Domain that is locally unique to the publisher process. The publisher processes use the Observation Domain ID to uniquely identify the collecting process of the Observation Domain that meters the metrics. Receivers SHOULD use the transport session and the Observation Domain ID field to separate different publisher streams originating from the same publisher.

3. Motivation

Lost and corrupt YANG notification messages need to be recognized at the receiver to ensure data integrity even when multiple publisher processes publishing from the same transport session.

To preserve data integrity down to the publisher process, the Observation Domain ID in the transport message header of the YANG notification message is introduced. In case of UDP transport, this is described in Section 3.2 of UDP based transport [I-D.ietf-netconf-udp-notif].

4. Solution Overview

Figure 2 below shows the distributed data export framework.

A collector usually includes two components,

* the Subscriber generates the subscription instructions to express what and how the Receiver want to receive the data;

* the Receiver is the target for the data publication.

For one subscription, there can be one or more Receivers. And the Subscriber does not necessarily share the same IP address as the Receivers.

In this framework, the Publisher pushes data to the Receiver according to the subscription. The Publisher is either in the Master or Agent role. The Master knows all the capabilities that his Agents can provide and exposes the Global Capability to the collector. The Subscriber maintains the Global Subscription at the Master and disassembles the Global Subscription to multiple Component Subscriptions, depending which source data is needed. The Component Subscriptions are then distributed to the corresponding Publisher Agents on route and processors on line cards.

Publisher Agents collects metrics according to the Component Subscription, add its metadata, encapsulates and pushes data to the Receiver where packets are reassembled and decapsulated.

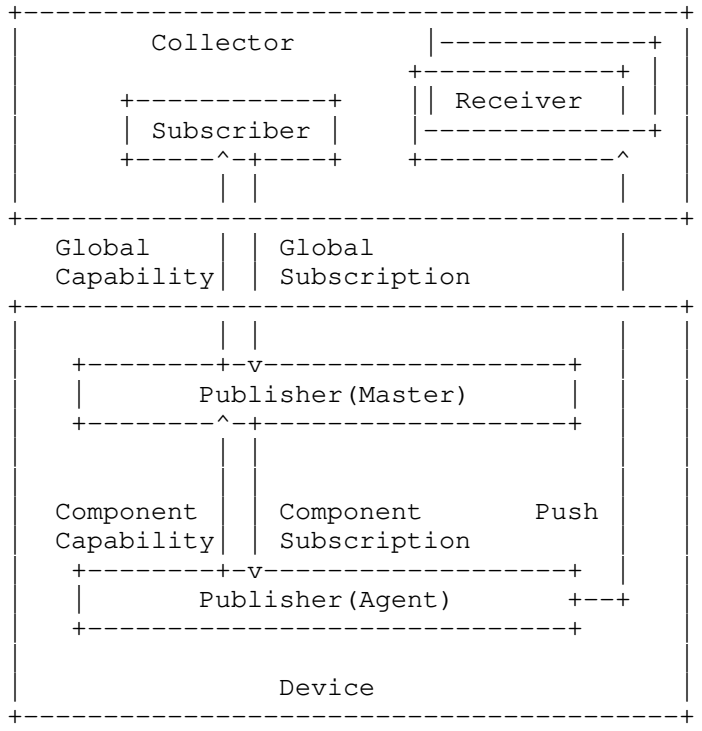

Figure 1: Fig. 2 The Distributed Data Export Framework

Master and Agents interact with each other in several ways:

* Agents need to register at the Master at the beginning of their process life-cycle

* Contracts are created between the Master and each Agent on the Component Capability, and the format for streaming data structure.

* The Master relays the component subscriptions to the Agents.

* The Agents announce the status of their Component Subscriptions to the Master. The status of the overall subscription is maintained by the Master. The Master is responsible for notifying the subscriber in case of problems with the Component Subscriptions.

The technical mechanisms or protocols used for the coordination of operational information between Master and Agent is out-of-scope of this document.

5. Subscription Decomposition

The Collector can only subscribe to the Master. This requires the Master to:

1. expose the Global Capability that can be served by multiple Publisher Agents;

2. disassemble the Global Subscription to multiple Component Subscriptions, and distribute them to the Publisher Agents of the corresponding metric sources so that they not overlap;

3. notify on changes when portions of a subscription moving between different Publisher Agents over time.

And the Agent to:

* Inherit the Global Subscription properties from Publisher Master for its Component Subscription;

* share the same life-cycle as the Global Subscription;

* share the same Subscription ID as the Global Subscription.

6. Publication Composition

The Publisher Agent collects data and encapsulates the packets per Component Subscription. The format and structure of the data records are defined by the YANG schema, so that the decomposition at the Receiver can benefit from the structured and hierarchical data records.

The Receiver is able to associate the YANG data records with Subscription ID [RFC8639] to the subscribed subscription and with Message Observation Domain ID [I-D.ietf-netconf-notification-messages] to one of the Publisher Agents software processes to enable message integrity.

For the dynamic subscription, the output of the "establish-subscription" RPC defined in [RFC8639] MUST include a list of Message Observation Domain IDs to indicate how the Global Subscription is decomposed into several Component Subscriptions.

The "subscription-started" and "subscription-modified" notification defined in [RFC8639] MUST also include a list of Message Observation Domain IDs to notify the current Publishers for the corresponding Global Subscription.

7. Subscription State Change Notifications

In addition to sending event records to Receivers, the Master MUST also send subscription state change notifications [RFC8639] when events related to subscription management have occurred. All the subscription state change notifications MUST be delivered by the Master.

When the subscription decomposition result changed, the "subscription-modified" notification MUST be sent to indicate the new list of Publishers.

8. Publisher Configurations

This document assumes that all Publisher Agents are preconfigured to push data. The actual working Publisher Agents are selected based on the subscription decomposition result.

All Publisher Agents share the same source IP address for data export. For connectionless data transport such as UDP based transport [I-D.ietf-netconf-udp-notif] the same Layer 4 source port for data export can be used. For connection based data transport such as HTTPS based transport [I-D.ietf-netconf-https-notif], each Publisher Agent MUST be able to acknowledge packet retrieval from Receivers, and therefore requires a dedicated Layer 4 source port per software process.

The specific configuration on transports is described in the responsible documents.

9. YANG Tree

```
module: ietf-distributed-notif
  augment /sn:subscriptions/sn:subscription:
    +--ro message-observation-domain-id*   string
  augment /sn:subscription-started:
    +--ro message-observation-domain-id*   string
  augment /sn:subscription-modified:
    +--ro message-observation-domain-id*   string
  augment /sn:establish-subscription/sn:output:
    +--ro message-observation-domain-id*   string
```

10. YANG Module

```
<CODE BEGINS> file "ietf-distributed-notif@2021-05-07.yang"
module ietf-distributed-notif {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-distributed-notif";
  prefix dn;
  import ietf-subscribed-notifications {
    prefix sn;
  }

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

     Editor:   Tianran Zhou
               <mailto:zhoutianran@huawei.com>

     Editor:   Guangying Zheng
               <mailto:zhengguangying@huawei.com>";


  description
    "Defines augmentation for ietf-subscribed-notifications to
    enable the distributed publication with single subscription.

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

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject to
    the license terms contained in, the Simplified BSD License set
```

```
      forth in Section 4.c of the IETF Trust's Legal Provisions
      Relating to IETF Documents
      (https://trustee.ietf.org/license-info).

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

    revision 2021-05-07 {
      description
        "Initial version";
      reference
        "RFC XXXX: Subscription to Distributed Notifications";
    }

    grouping message-observation-domain-ids {
      description
        "Provides a reusable list of message-observation-domain-ids.";

      leaf-list message-observation-domain-id {
        type string;
        config false;
        ordered-by user;
        description
          "Software process which created the message (e.g.,
           processor 1 on line card 1). This field is
           used to  notify the collector the working originator.";
      }
    }

    augment "/sn:subscriptions/sn:subscription" {
      description
        "This augmentation allows the message
        Observation Domain ID to be exposed for a subscription.";

      uses message-observation-domain-ids;
    }

    augment "/sn:subscription-started" {
      description
        "This augmentation allows MSO specific parameters to be
         exposed for a subscription.";

      uses message-observation-domain-ids;
    }

    augment "/sn:subscription-modified" {
      description
        "This augmentation allows MSO specific parameters to be
```

```
          exposed for a subscription.";

      uses message-observation-domain-ids;
    }

    augment "/sn:establish-subscription/sn:output" {
      description
        "This augmentation allows MSO specific parameters to be
         exposed for a subscription.";

      uses message-observation-domain-ids;
    }
  }
  <CODE ENDS>
```

11. IANA Considerations

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

URI: urn:ietf:params:xml:ns:yang:ietf-distributed-notif

Registrant Contact: The IESG.

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

This document registers the following YANG module in the YANG Module Names registry [RFC3688]:

Name: ietf-distributed-notif

Namespace: urn:ietf:params:xml:ns:yang:ietf-distributed-notif

Prefix: dn

Reference: RFC XXXX

12. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF Access Control Model (NACM) [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The new data nodes introduced 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-config or notification) to this data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

* /subscriptions/subscription/message-observation-domain-ids

The entries in the two lists above will show where subscribed resources might be located on the publishers. Access control MUST be set so that only someone with proper access permissions has the ability to access this resource.

Other Security Considerations is the same as those discussed in [RFC8639].

13. Contributors

Alexander Clemm
Futurewai
2330 Central Expressway
Santa Clara
California
United States of America
Email: ludwig@clemm.org

14. Acknowledgements

We thank Kent Watsen, Mahesh Jethanandani, Martin Bjorklund, Tim Carey and Qin Wu for their constructive suggestions for improving this document.

15. References

15.1. 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>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

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

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

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

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <https://www.rfc-editor.org/info/rfc7923>.

[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>.

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

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

15.2. Informative References

[I-D.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-10, 15 June 2022, <https://www.ietf.org/archive/id/draft-ietf-netconf-https-notif-10.txt>.

[I-D.ietf-netconf-notification-messages]
Voit, E., Jenkins, T., Birkholz, H., Bierman, A., and A. Clemm, "Notification Message Headers and Bundles", Work in Progress, Internet-Draft, draft-ietf-netconf-notification-messages-08, 17 November 2019, <https://www.ietf.org/archive/id/draft-ietf-netconf-notification-messages-08.txt>.

[I-D.ietf-netconf-udp-notif]
Zhou, T., Zheng, G., Lucente, P., Graf, T., and P. Francois, "UDP-based Transport for Configured Subscriptions", Work in Progress, Internet-Draft, draft-ietf-netconf-udp-notif-01, July 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-udp-notif-01>.

Appendix A. Examples

This appendix is non-normative.

A.1. Dynamic Subscription

Figure 3 shows a typical dynamic subscription to the device with distributed data export capability.

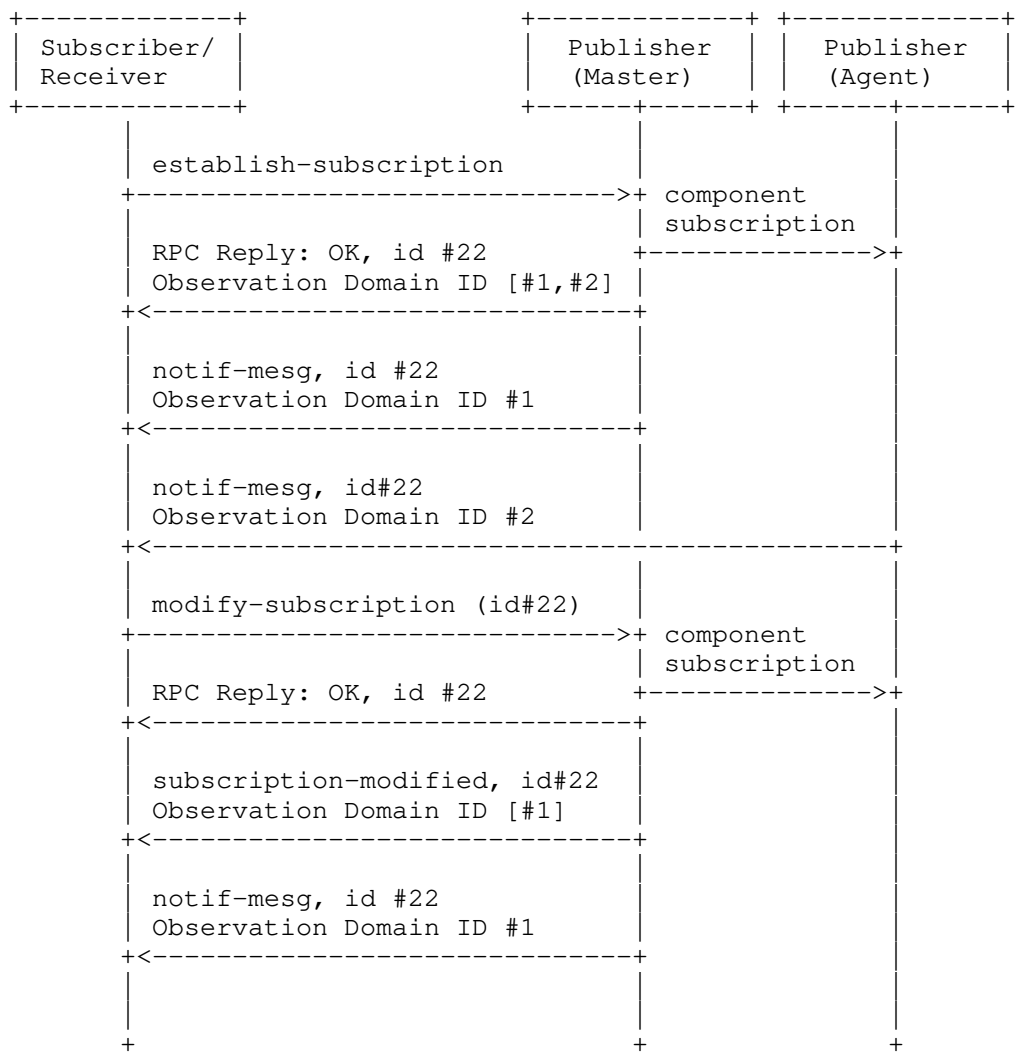

Figure 2: Fig. 3 Call Flow for Dynamic Subscription

A "establish-subscription" RPC request as per [RFC8641] is sent to the Master with a successful response. An example of using NETCONF:

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

Figure 3: Fig. 4 "establish-subscription" Request

As the device is able to fully satisfy the request, the request is given a subscription ID of 22. The response as in Figure 5 indicates that the subscription is decomposed into two component subscriptions which will be published by two message Observation Domain ID: #1 and #2.

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <id
   xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
     22
  </id>
  <message-observation-domain-id
   xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
     1
  </message-observation-domain-id>
  <message-observation-domain-id
   xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
     2
  </message-observation-domain-id>
</rpc-reply>
```

Figure 4: Fig. 5 "establish-subscription" Positive RPC Response

Then, both Publishers send notifications with the corresponding piece of data to the Receiver.

The subscriber may invoke the "modify-subscription" RPC for a subscription it previously established. The RPC has no difference to the single publisher case as in [RFC8641]. Figure 6 provides an example where a subscriber attempts to modify the period and datastore XPath filter of a subscription using NETCONF.

```
     <rpc message-id="102"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <modify-subscription
        xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
        xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
      <id>22</id>
      <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:bar
      </yp:datastore-xpath-filter>
      <yp:periodic>
        <yp:period>250</yp:period>
      </yp:periodic>
     </modify-subscription>
  </rpc>
```

Figure 5: Fig. 6 "modify-subscription" Request

If the modification is successfully accepted, the "subscription-modified" subscription state notification is sent to the subscriber by the Master. The notification, Figure 7 for example, indicates the modified subscription is decomposed into one component subscription which will be published by message Observation Domain #1.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
<eventTime>2007-09-01T10:00:00Z</eventTime>
<subscription-modified
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
      xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>22</id>
    <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:bar
    </yp:datastore-xpath-filter>
    <yp:periodic>
        <yp:period>250</yp:period>
    </yp:periodic>
    <message-observation-domain-id
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notificationss>
       1
    </message-observation-domain-id>
  </subscription-modified>
</notification>
```

Figure 6: Fig. 7 "subscription-modified" Subscription State Notification

A.2. Configured Subscription

Figure 8 shows a typical configured subscription to the device with distributed data export capability.

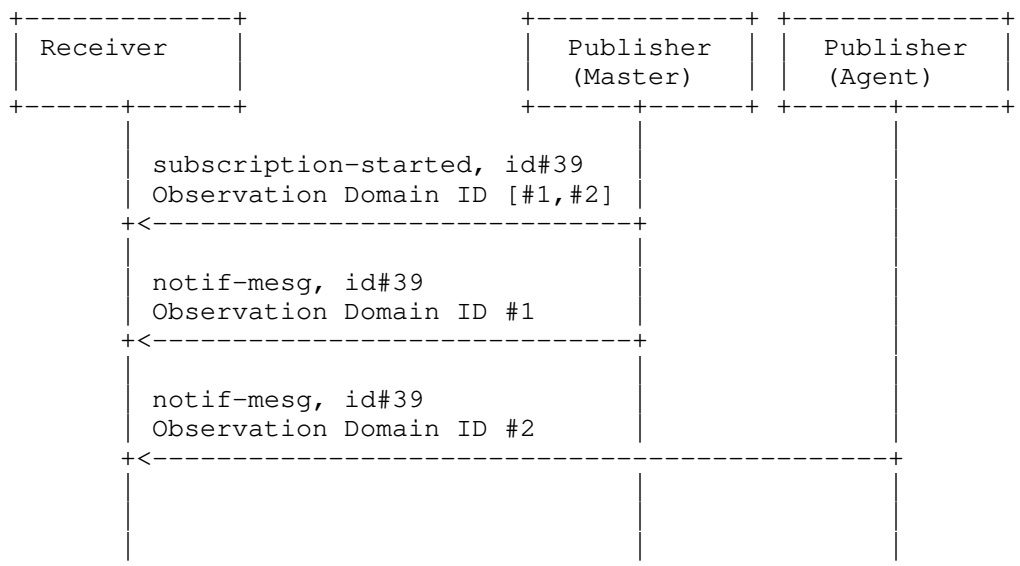

Figure 7: Fig. 8 Call Flow for Configured Subscription

Before starting to push data, the "subscription-started" subscription state notification is sent to the Receiver. The following example assumes the NETCONF transport has already established. The notification indicates that the configured subscription is decomposed into two component subscriptions which will be published by two message Observation Domain: #1 and #2.

```
 <notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2007-09-01T10:00:00Z</eventTime>
   <subscription-started
       xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
     <identifier>39</identifier>
     <yp:datastore
         xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
       ds:operational
     </yp:datastore>
     <yp:datastore-xpath-filter
         xmlns:ex="https://example.com/sample-data/1.0">
       /ex:foo
     </yp:datastore-xpath-filter>
     <yp:periodic>
         <yp:period>250</yp:period>
     </yp:periodic>
     <message-observation-domain-id
     xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
        1
     </message-observation-domain-id>
     <message-observation-domain-id
     xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
        2
     </message-observation-domain-id>
   </subscription-started>
 </notification>
```

Figure 8: Fig. 9 "subscription-started" Subscription State Notification

Then, both Publishers send notifications with the corresponding data record to the Receiver.

Authors' Addresses

Tianran Zhou
Huawei
156 Beiqing Rd., Haidian District
Beijing
China
Email: zhoutianran@huawei.com

Guangying Zheng
Huawei
101 Yu-Hua-Tai Software Road
Nanjing
Jiangsu,
China
Email: zhengguangying@huawei.com

Eric Voit
Cisco Systems
United States of America
Email: evoit@cisco.com

Thomas Graf
Swisscom
Binzring 17
CH- Zuerich 8045
Switzerland
Email: thomas.graf@swisscom.com

Pierre Francois
INSA-Lyon
Lyon
France
Email: pierre.francois@insa-lyon.fr

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 January 2023

K. Watsen
Watsen Networks
Q. Wu
Huawei Technologies
O. Hagsand
Netgate
H. Li
Hewlett Packard Enterprise
P. Andersson
Cisco Systems
24 July 2022

List Pagination for YANG-driven Protocols
draft-ietf-netconf-list-pagination-00

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 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]. Details for how such protocols are updated are outside the scope of this document.

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 used in this document use a placeholder value for binary data that has been base64 encoded (e.g., "BASE64VALUE="). This placeholder value is used as 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 only defines a YANG extension and augments a couple 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.

A secondary aspect of this solution entails a client sending a query parameter to a server guiding how descendent 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 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 the "direction" parameter (see Section 3.1.3), then the "offset" parameter (see Section 3.1.4), and lastly the "limit" parameter (see Section 3.1.5).

3.1.1. The "where" Query Parameter

Description
: The "where" query parameter specifies a filter expression that result-set entries must match.

Default Value
: If this query parameter is unspecified, then no entries are filtered from the working result-set.

Allowed Values
: The allowed values are XPath 1.0 expressions. It is an error if the XPath expression references a node identifier that does not exist in the schema, is optional or conditional in the schema or, for constrained "config false" lists and leaf-lists (see Section 3.3), if the node identifier does not point to a node having the "indexed" extension statement applied to it (see Section 3.3.2).

Conformance
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. 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. It is an error if the specified node identifier does not exist in the schema, is optional or conditional in the schema or, for constrained "config false" lists and leaf-lists (see Section 3.3), if the node identifier does not point to a node having the "indexed" extension statement applied to it (see Section 3.3.2).

Conformance
The "sort-by" query parameter MUST be supported for all "config true" lists and leaf-lists and 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.

3.1.3. 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

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

3.1.4. 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.

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

Allowed Values
The allowed values are unsigned integers. It is an error for the offset value to exceed the number of entries in the working result-set, and the "offset-out-of-range" identity SHOULD be produced in the error output when this occurs.

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

3.1.5. 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.

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

Allowed Values
: The allowed values are positive integers.

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 descendent lists and leaf-lists.

  Any descendent 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 descendent lists and leaf-lists is unbounded.

Allowed Values
: The allowed values are positive integers.

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, 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 already servers must 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 [I-D.ietf-netconf-notification-capabilities].

Specifically, the "ietf-list-pagination" module (see Section 4) augments an empty 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 XPath 1.0 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 applied to it (see Section 3.3.2).

* For lists only, node-identifiers used in "where" expressions and "sort-by" filters MUST NOT descend past any descendent 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 an empty leaf node called "indexed" into the "per-node-capabilities" node defined in the "ietf-system-capabilities" module (see [I-D.ietf-netconf-notification-capabilities]).

When a "list" or "leaf-list" node has the "constrained" leaf, only nodes having the "indexed" node may be used in "where" and/or "sort-by" expressions. If no nodes have the "indexed" leaf, when the "constrained" leaf is present, 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?   empty
    +--ro indexed?       empty
```

Comments:

* As shown, this module augments two optional leaves into the "node-selector" 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 3.1.5) or "sublist-limit" parameters (Appendix A.3.6).

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="http://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/>
       </per-node-capabilities>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log/es:timestamp</node-\
   selector>
         <lpg:indexed/>
       </per-node-capabilities>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log/es:member-id</node-\
   selector>
         <lpg:indexed/>
       </per-node-capabilities>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log/es:outcome</node-se\
   lector>
         <lpg:indexed/>
       </per-node-capabilities>
     </datastore-capabilities>
   </system-capabilities>
```

4.2.2. Indicating number remaining in a limited list

FIXME: valid syntax for 'where'?

4.3. YANG Module

This YANG module has normative references to [RFC7952] and [I-D.ietf-netconf-notification-capabilities].

```
   <CODE BEGINS> file "ietf-list-pagination@2022-07-24.yang"

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

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

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

  import ietf-system-capabilities  {
    prefix sysc;
    reference
      "draft-ietf-netconf-notification-capabilities:
       YANG Modules describing Capabilities for
       Systems and Datastore Update Notifications";
  }

  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>";

  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) 2022 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

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

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

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

    revision 2022-07-24 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: List Pagination for YANG-driven Protocols";
    }


    // Annotations

    md:annotation remaining {
      type union {
        type uint32;
        type enumeration {
          enum "unknown" {
            description
              "Indicates that 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.";
    }

    // 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.";
    }

    // Groupings

    grouping where-param-grouping {
      description
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      leaf 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.

           It is an error if the XPath expression references a node
           identifier that does not exist in the schema, is optional
           or conditional in the schema or, for constrained 'config
           false' lists and leaf-lists, if the node identifier does
           not point to a node having the 'indexed' extension
           statement applied to it (see RFC XXXX).";
      }
    }

    grouping sort-by-param-grouping {
      description
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      leaf sort-by {
        type union {
          type string {
            // An RFC 7950 'descendant-schema-nodeid'.
            pattern '([0-9a-fA-F]*:)?[0-9a-fA-F]*'
                    + '(/([0-9a-fA-F]*:)?[0-9a-fA-F]*)*';
          }
```

```
          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).";
      }
    }

    grouping direction-param-grouping {
      description
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      leaf direction {
        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.";
      }
    }

    grouping offset-param-grouping {
      description
```

```
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      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 over
           when preparing the response.";
      }
    }

    grouping limit-param-grouping {
      description
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      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.";
      }
    }

    grouping sublist-limit-param-grouping {
      description
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      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 descendent 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 // FIXME: ensure datastore == <operational>
      "/sysc:system-capabilities/sysc:datastore-capabilities"
      + "/sysc:per-node-capabilities" {
      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 empty;
        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'.";
      }
      leaf indexed {
        type empty;
        description
          "Indicates that the targeted  descendent node of a
           'constrained' list (see the 'constrained' leaf) may be
           used in 'where' filters and/or 'sort-by' parameters.
           If a descendent node of a 'constrained' list is not
           'indexed', then it MUST NOT be used in 'where' filters
```

```
            or 'sort-by' parameters.";
        }
      }
    }
```

<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. Security Considerations

6.1. Regarding the "ietf-list-pagination" YANG Module

Pursuant the template defined in ...FIXME

7. References

7.1. Normative References

[I-D.ietf-netconf-notification-capabilities]
Lengyel, B., Clemm, A., and B. Claise, "YANG Modules Describing Capabilities for Systems and Datastore Update Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-notification-capabilities-21, 15 October 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-notification-capabilities-21>.

[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>.

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

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

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

7.2. Informative References

[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>.

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

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

[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>.

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 pecularities 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 "http://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 visable 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 members 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 uint64 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@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"
          }
        },
        {
          "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: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.

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.

> If '0' were a valid limit value, it would always return an empty result set. 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
```

A.3.3. 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.3.3.1. 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.3.3.2. 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.3.4. 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.3.4.1. 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 descendent and the other using an indirect descendent.

A.3.4.1.1. 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.3.4.1.2. type is a "list" and sort-by node is a direct descendent

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

This vector test uses the target "/example-social:members/member", which is a "list", and the sort-by descendent 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.3.4.1.3. type is a "list" and sort-by node is an indirect descendent

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

This vector test uses the target "/example-social:members/member", which is a "list", and the sort-by descendent node "stats/joined", which is a "config false" descendent 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.3.4.2. handling missing entries

The section provides one example for when the "sort-by" node is not present in the data set.

FIXME: need to finish this section...

A.3.5. The "where" Parameter

The "where" is an 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.3.5.1. match of leaf-list’s values

FIXME

A.3.5.2. match on descendent 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",
      ...
    },
    {
      "member-id": "lin",
      ...
    }
  ]
}
```

A.3.5.3. 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.3.6. The "sublist-limit" Parameter

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

A.3.6.1. 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 descendent lists and leaf-lists.

Note that, in the response, the "remaining" metadata value is set on the first element of each descendent 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.3.6.2. target is a datastore

This example uses the target node <intended>.

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

Note that, in the response, the "remaining" metadata value is set on the first element of each descendent 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.3.7. Combinations of Parameters

A.3.7.1. 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

The authors would like to thank the following for lively discussions on list (ordered by first name): Andy Bierman, Martin Bjoerklund, and Robert Varga.

Authors' Addresses

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

Qin Wu
Huawei Technologies

Email: bill.wu@huawei.com

Olof Hagsand
Netgate
Email: olof@hagsand.se

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

Per Andersson
Cisco Systems
Email: perander@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 January 2023

K. Watsen
Watsen Networks
Q. Wu
Huawei
O. Hagsand
Netgate
H. Li
HPE
P. Andersson
Cisco Systems
24 July 2022

NETCONF Extensions to Support List Pagination
draft-ietf-netconf-list-pagination-nc-00

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.

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 used in this document use a placeholder value for binary data that has been base64 encoded (e.g., "BASE64VALUE="). This placeholder value is used as 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, similar to the "filter" parameter of <get-config>, but includes nodes for "list" and "leaf-list" filtering.

The content filter 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.

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?           union
       +---w sort-by?         union
       +---w direction?       enumeration
       +---w offset?          uint32
       +---w limit?           union
       +---w sublist-limit?   union
  augment /nc:get-config/nc:input:
    +---w list-pagination
       +---w where?           union
       +---w sort-by?         union
       +---w direction?       enumeration
       +---w offset?          uint32
       +---w limit?           union
       +---w sublist-limit?   union
  augment /ncds:get-data/ncds:input:
    +---w list-pagination
       +---w where?           union
       +---w sort-by?         union
       +---w direction?       enumeration
       +---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 then the number of instances in the list or leaf-list target resource, 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@2022-07-24.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 lp;
    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:  NETCONF WG list <mailto:netconf@ietf.org>";
```

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

     Copyright (c) 2021 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the 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 2022-07-24 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: NETCONF Extensions to Support List Pagination";
  }

  grouping pagination-parameters {
    description "A grouping for list pagination parameters.";
    container list-pagination {
      description "List pagination parameters.";
      uses lp:where-param-grouping;
      uses lp:sort-by-param-grouping;
      uses lp:direction-param-grouping;
      uses lp:offset-param-grouping;
      uses lp:limit-param-grouping;
      uses lp:sublist-limit-param-grouping;
    }
  }

  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 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 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 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

The YANG module defined in this document extends the base operations for NETCONF [RFC6241] and 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] apply to the new <get-list-pagination> RPC operations defined in this document.

8. References

8.1. Normative References

[I-D.ietf-netconf-list-pagination]
"List Pagination...", <FIXME>.

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

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

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

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

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

[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>.

[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>.

[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>.

8.2. Informative References

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

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

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

Appendix A. Open Issues

Cursors (i.e.,stable result sets) are related to the topic of dynamic changing lists between two queries. How cursors can be supported using "feature"?

Appendix B. 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 C. 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 D. Example Queries

D.1. List pagination with all query parameters

This example mimics that Appendix A.3.7 of [I-D.ietf-netconf-list-pagination].

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

<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="42">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="xpath" select="/es:members/es:member"
      xmlns:es="http://example.com/ns/example-social"/>
      <list-pagination
        xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-list-paginat\
ion">true</list-pagination>
      <where>//stats//joined[starts-with(@timestamp,'2020')]</where>
      <sort-by>timestamp</sort-by>
      <direction>backwards</direction>
      <offset>2</offset>
      <limit>2</limit>
      <sublist-limit>1</sublist-limit>
    </filter>
  </get-config>
</rpc>
```

Response from the NETCONF server:

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

<lp:xml-list xmlns:lp="urn:ietf:params:xml:ns:yang:ietf-restconf-lis\
t-pagination"
  xmlns="http://example.com/ns/example-social">
  <member lp:remaining="1">
    <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 a 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">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">
    <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">
        <timestamp>2020-08-14T03:32:25Z</timestamp>
        <body>Just got in.</body>
      </post>
    </posts>
    <favorites>
      <decimal64-numbers lp:remaining="1">3.14159</bits>
    </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>
  </lp:xml-list>
```

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:

Andy Bierman Martin Björklund Robert Varga

Authors' Addresses

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

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com

Olof Hagsand
Netgate
Email: olof@hagsand.se

Hongwei Li
HPE
Email: flycoolman@gmail.com

Per Andersson
Cisco Systems
Email: perander@cisco.com

NETCONF Working Group
Internet-Draft
Updates: 8040 (if approved)
Intended status: Standards Track
Expires: 25 January 2023

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

24 July 2022

RESTCONF Extensions to Support List Pagination
draft-ietf-netconf-list-pagination-rc-00

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 and DELETE operations, 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. Declaring these nodes as valid resource targets for the DELETE operation merely completes the solution for RESTCONF.

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 used in this document use a placeholder value for binary data that has been base64 encoded (e.g., "BASE64VALUE="). This placeholder value is used as 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" and "DELETE" operations.

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", "direction", "sort-by", "where", and "sublist-list".

These six query parameters correspond to those defined in Sections 3.1 and 3.2 in [I-D.ietf-netconf-list-pagination].

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 the skipped over when preparing the response. If not specified, then no entries in the result set are skipped.
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.
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 lists and leaf-lists, but servers 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 3.1.5 of [I-D.ietf-netconf-list-pagination].

If the 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".

2.3.2. The "offset" Query Parameter

The "offset" query parameter corresponds to the "offset" parameter defined in Section 3.1.4 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 error-app-tag value.

2.3.3. The "direction" Query Parameter

The "direction" query parameter corresponds to the "direction" parameter defined in Section 3.1.3 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".

2.3.4. 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".

2.3.5. 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].

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".

2.3.6. 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 sumlist-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 the 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
:direction	urn:ietf:params:restconf:capability:direction:1.0
:sort-by	urn:ietf:params:restconf:capability:sort-by: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

[I-D.ietf-netconf-list-pagination]
"List Pagination...", <FIXME>.

[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>.

5.2. Informative References

[I-D.ietf-netconf-restconf-collection] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Collection Resource", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-collection-00, 30 January 2015, <https://www.ietf.org/archive/id/draft-ietf-netconf-restconf-collection-00.txt>.

[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>.

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.3.7 of [I-D.ietf-netconf-list-pagination].

```
   ============== 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=tim\
   estamp&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

   <lp:xml-list xmlns:lp="urn:ietf:params:xml:ns:yang:ietf-restconf-lis\
   t-pagination"
     xmlns="http://example.com/ns/example-social">
     <member lp:remaining="1">
       <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 a 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">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">
       <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">
          <timestamp>2020-08-14T03:32:25Z</timestamp>
          <body>Just got in.</body>
        </post>
      </posts>
      <favorites>
        <decimal64-numbers lp:remaining="1">3.14159</bits>
      </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>
  </lp:xml-list>
```

C.2. Deletion of a leaf-list

This example illustrates using a "leaf-list" as the DELETE target.

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

DELETE /restconf/ds/ietf-datastores:running/example-social:members/m\
ember=bob/favorites/decimal64-numbers HTTP/1.1
Host: example.com
Accept: application/yang-data+xml

Response from the RESTCONF server:

HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
```

Acknowledgements

This work has benefited from the discussions of restconf resource collection over the years, in particular, [I-D.ietf-netconf-restconf-collection]. The authors additionally thank the following for lively discussions on list (ordered by first name): Andy Bierman, Martin Bjoerklund, and Robert Varga

Authors' Addresses

Kent Watsen
Watsen Networks

Email: kent+ietf@watsen.net

Qin Wu
Huawei Technologies
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com

Olof Hagsand
Netgate
Email: olof@hagsand.se

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

Per Andersson
Cisco Systems
Email: perander@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 10 December 2022

J. Lindblad
Cisco Systems

8 June 2022

Transaction ID Mechanism for NETCONF
draft-lindblad-netconf-transaction-id-02

Abstract

NETCONF clients and servers often need to have a synchronized view of the server's configuration data stores. The volume of configuration data in a server may be very large, while data store changes typically are small when observed at typical client resynchronization intervals.

Rereading the entire data store and analyzing the response for changes is an inefficient mechanism for synchronization. This document specifies an extension to NETCONF that allows clients and servers to keep synchronized with a much smaller data exchange and without any need for servers to store information about the clients.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/netconf-wg/netconf-etag.

Status of This Memo

Table of Contents

1. Introduction

When a NETCONF client connects with a NETCONF server, a frequently occurring use case is for the client to find out if the configuration has changed since it was last connected. Such changes could occur for example if another NETCONF client has made changes, or another system or operator made changes through other means than NETCONF.

One way of detecting a change for a client would be to retrieve the entire configuration from the server, then compare the result with a previously stored copy at the client side. This approach is not popular with most NETCONF users, however, since it would often be very expensive in terms of communications and computation cost.

Furthermore, even if the configuration is reported to be unchanged, that will not guarantee that the configuration remains unchanged when a client sends a subsequent change request, a few moments later.

In order to simplify the task of tracking changes, a NETCONF server could implement a meta level checksum over the configuration over a datastore or YANG subtree, and offer clients a way to read and compare this checksum. If the checksum is unchanged, clients can avoid performing expensive operations. Such checksums are often referred to as a configuration id or transaction id (txid).

Evidence of a transaction id feature being demanded by clients is that several server implementors have built proprietary and mutually incompatible mechanisms for obtaining a transaction id from a NETCONF server.

RESTCONF, RFC 8040 (https://tools.ietf.org/html/rfc8040), defines a mechanism for detecting changes in configuration subtrees based on Entity-Tags (ETags) and Last-Modified txid values.

In conjunction with this, RESTCONF provides a way to make configuration changes conditional on the server confiuguration being untouched by others. This mechanism leverages RFC 7232 (https://tools.ietf.org/html/rfc7232) "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests".

This document defines similar functionality for NETCONF, RFC 6241 (https://tools.ietf.org/html/rfc6241), and ties this in with YANG-Push, RFC 8641 (https://tools.ietf.org/html/rfc8641).

2. Conventions and Definitions

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

This document uses the terminology defined in RFC6241 (https://tools.ietf.org/html/rfc6241), RFC7950 (https://tools.ietf.org/html/rfc7950), RFC8040 (https://tools.ietf.org/html/rfc8040), and RFC8641 (https://tools.ietf.org/html/rfc8641).

In addition, this document defines the following terms:

Versioned node A node in the instantiated YANG data tree for which the server maintains a transaction id (txid) value.

3. NETCONF Txid Extension

This document describes a NETCONF extension which modifies the behavior of get-config, get-data, edit-config, edit-data, discard-changes, copy-config, delete-config and commit such that clients are able to conditionally retrieve and update the configuration in a NETCONF server.

For servers implementing YANG-Push, an extension for conveying txid updates as part of subscription updates is also defined.

Several low level mechanisms could be defined to fulfill the requirements for efficient client-server txid synchronization. This document defines two such mechanisms, the etag txid mechanism and the last-modified txid mechanism. Additional mechanisms could be added in future.

3.1. Use Cases

The common use cases for such mecahnisms are briefly discussed here.

Initial configuration retrieval When the client initially connects

to a server, it may be interested to acquire a current view of (parts of) the server's configuration. In order to be able to efficiently detect changes later, it may also be interested to store meta level txid information for subtrees of the configuration.

Subsequent configuration retrieval When a client needs to reread (parts of) the server's configuration, it may be interested to leverage the txid meta data it has stored by requesting the server to prune the response so that it does not repeat configuration data that the client is already aware of.

Configuration update with txid return When a client issues a transaction towards a server, it may be interested to also learn the new txid meta data the server has stored for the updated parts of the configuration.

Configuration update with txid specification When a client issues a transaction towards a server, it may be interested to also specify the new txid meta data that the server stores for the updated parts of the configuration.

Conditional configuration change When a client issues a transaction towards a server, it may specify txid meta data for the transaction in order to allow the server to verify that the client is up to date with any changes in the parts of the configuration that it is concerned with. If the txid meta data in the server is different than the client expected, the server rejects the transaction with a specific error message.

Subscribe to configuration changes with txid return When a client subscribes to configuration change updates through YANG-Push, it may be interested to also learn the the updated txid meta data for the changed data trees.

3.2. General Txid Principles

All servers implementing a txid mechanism MUST maintain a txid meta data value for each configuration datastore supported by the server. Txid mechanism implementations MAY also maintain txid meta data values for nodes deeper in the YANG data tree. The nodes for which the server maintains txids are collectively referred to as the "versioned nodes".

The server returning txid values for the versioned nodes MUST ensure the txid values are changed every time there has been a configuration change at or below the node associated with the txid value. This means any update of a config true node will result in a new txid value for all ancestor versioned node, up to and including the datastore root itself.

This also means a server MUST update the txid value for any nodes that change as a result of a configuration change, regardless of source, even if the changed nodes are not explicitly part of the change payload. An example of this is dependent data under YANG RFC 7950 (https://tools.ietf.org/html/rfc7950) when- or choice-statements.

The server MUST NOT change the txid value of a versioned node unless the node itself or a child node of that node has been changed. The server MUST NOT change any txid values due to changes in config false data.

3.3. Initial Configuration Retrieval

When a NETCONF server receives a get-config or get-data request containing requests for txid values, it MUST return txid values for all versioned nodes below the point requested by the client in the reply.

The exact encoding varies by mechanism, but all txid mechanisms would have a special "txid-request" txid value (e.g. "?") which is guaranteed to never be used as a normal txid value. Clients MAY use this special txid value associated with one or more nodes in the data tree to indicate to the server that they are interested in txid values below that point of the data tree.

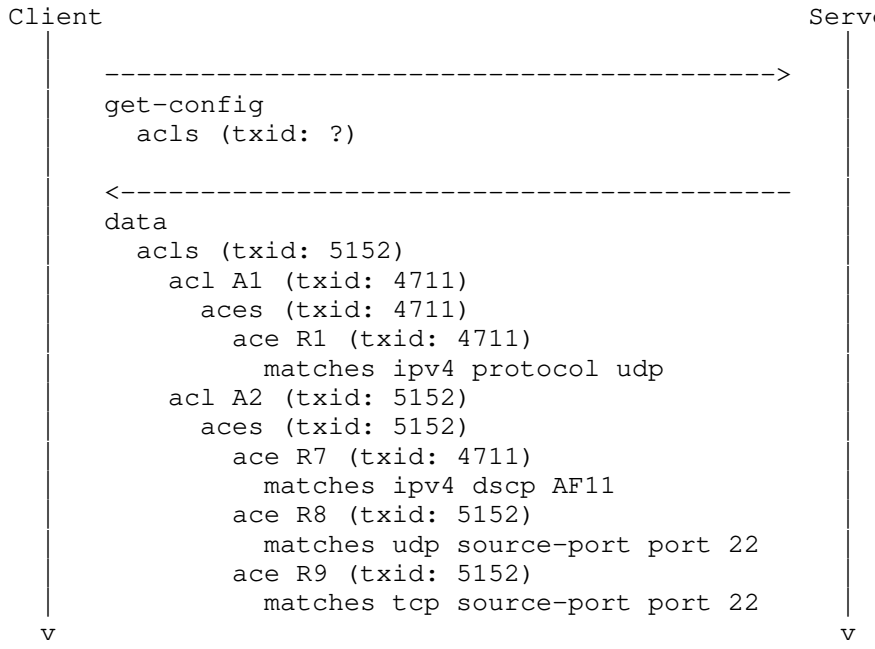

Figure 1: Initial Configuration Retrieval. The server returns the requested configuration, annotated with txid values. The most recent change seems to have been an update to the R8 and R9 source-port.

NOTE: In the call flow examples we are using a 4-digit, monotonously increasing integer as txid. This is convenient and enhances readability of the examples, but does not reflect a typical implementation. In general, the only operation defined on a pair of txid values is testing them for equality.

3.4. Subsequent Configuration Retrieval

Clients MAY request the server to return txid values in the response by adding one or more txid values received previously in get-config or get-data requests.

When a NETCONF server receives a get-config or get-data request containing a node with a client specified txid value, there are several different cases:

* The node is not a versioned node, i.e. the server does not maintain a txid value for this node. In this case, the server MUST look up the closest ancestor that is a versioned node, and use the txid value of that node as the txid value of this node in the further handling below. The datastore root is always a versioned node.

* The client specified txid value is different than the server's txid value for this node. In this case the server MUST return the contents as it would otherwise have done, adding the txid values of all child versioned nodes to the response. In case the client has specified txid values for some child nodes, then these cases MUST be re-evaluated for those child nodes.

* The node is a versioned node, and the client specified txid value matches the server's txid value. In this case the server MUST return the node decorated with a special "txid-match" txid value (e.g. "=") to the matching node, pruning any value and child nodes. A server MUST NOT ever use the txid-match value (e.g. "=") as an actual txid value.

For list elements, pruning child nodes means that top-level key nodes MUST be included in the response, and other child nodes MUST NOT be included. For containers, child nodes MUST NOT be included.

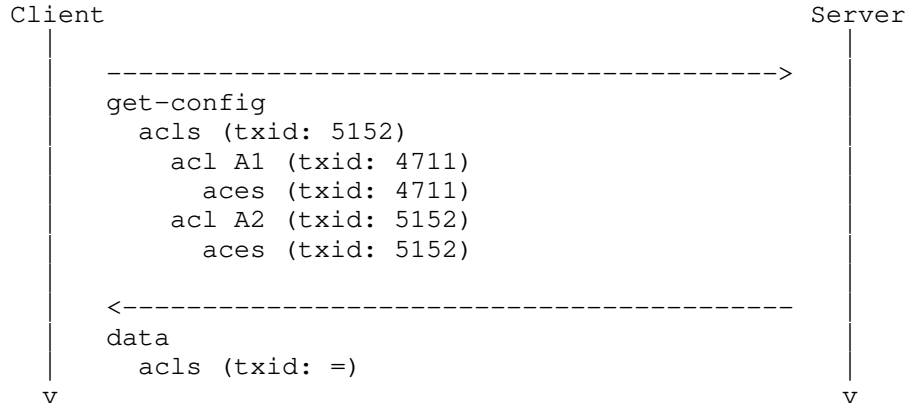

Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where txid matches expectations.

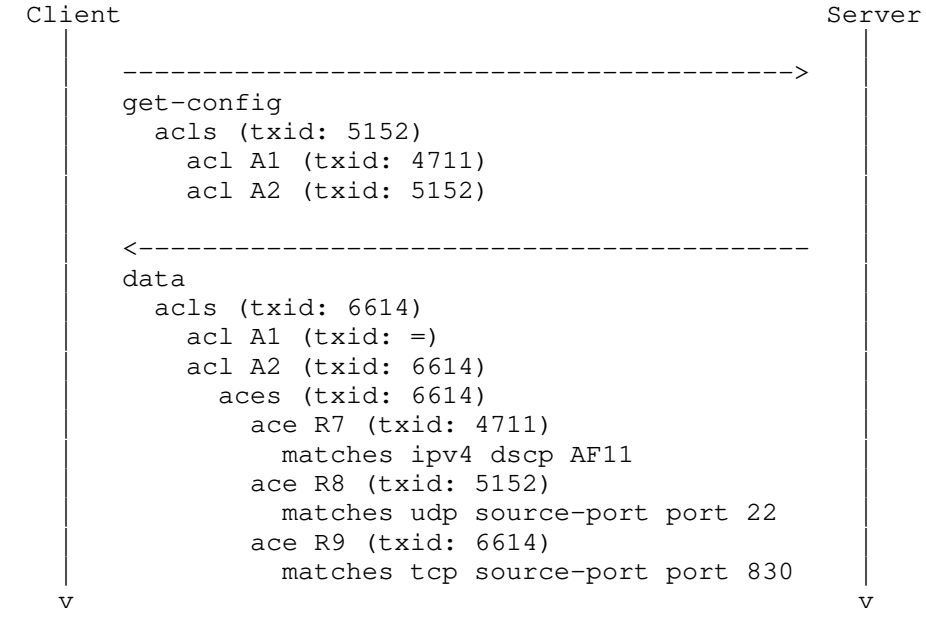

Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides update where changes have happened.

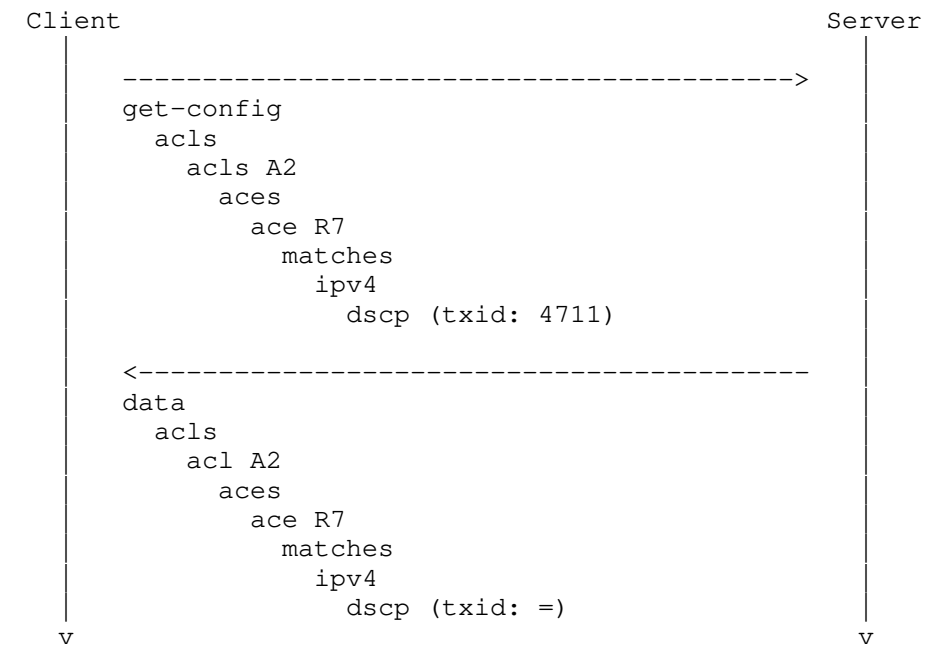

Figure 4: Versioned nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the etag value is '=', and the leaf value is pruned.

3.5. Conditional Transactions

Conditional transactions are useful when a client is interested to make a configuration change, being sure that relevant parts of the server configuration have not changed since the client last inspected it.

By supplying the latest txid values known to the client in its change requests (edit-config etc.), it can request the server to reject the transaction in case any relevant changes have occurred at the server that the client is not yet aware of.

This allows a client to reliably compute and send confiuguration changes to a server without either acquiring a global datastore lock for a potentially extended period of time, or risk that a change from another client disrupts the intent in the time window between a read (get-config etc.) and write (edit-config etc.) operation.

Clients that are also interested to know the txid assigned to the modified versioned nodes in the model immediately in the response could set a flag in the rpc message to request the server to return the new txid with the ok message.

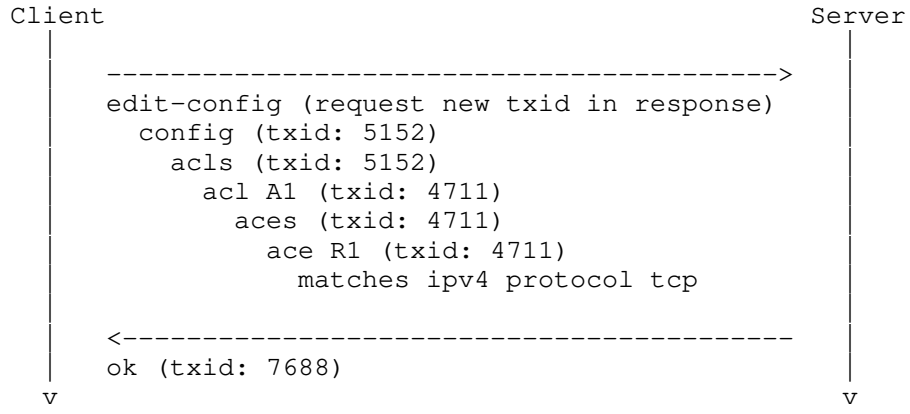

Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

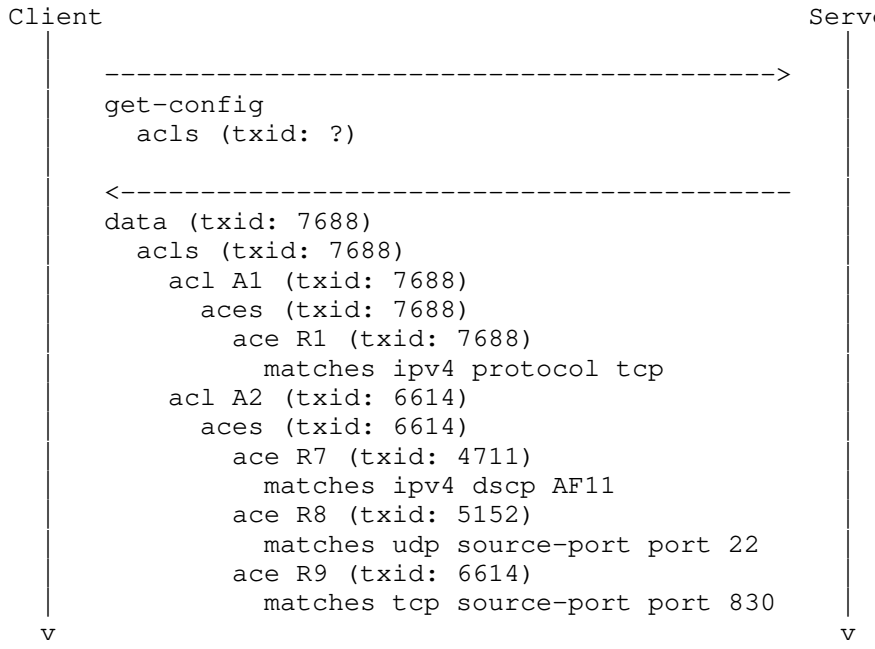

Figure 6: For all leaf objects that were changed, and all their ancestors, the txids are updated to the value returned in the ok message.

If the server rejects the transaction because the configuration txid value differs from the client's expectation, the server MUST return an rpc-error with the following values:

```
error-tag:      operation-failed
error-type:     protocol
error-severity: error
```

Additionally, the error-info tag SHOULD contain an sx:structure containing relevant details about the mismatching txids.

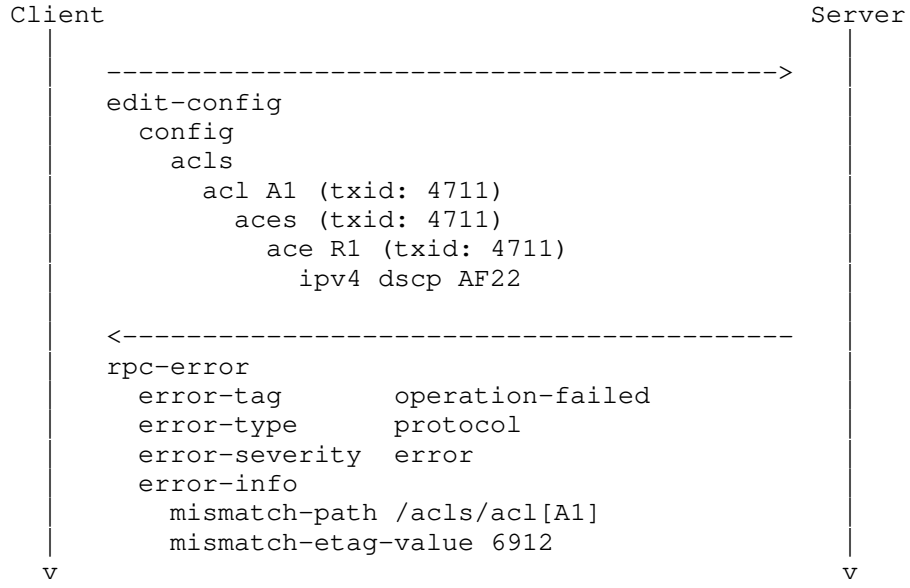

Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the txid it knows for this part of the configuration. Since the txid has changed (out of band), the server rejects the configuration change request and reports an error with details about where the mismatch was detected.

3.5.1. Transactions toward the Candidate Datastore

When working with the Candidate datastore, the txid validation happens at commit time, rather than at individual edit-config or edit-data operations. Clients add their txid attributes to the configuration payload the same way. In case a client specifies different txid values for the same element in successive edit-config or edit-data operations, the txid value specified last MUST be used by the server at commit time.

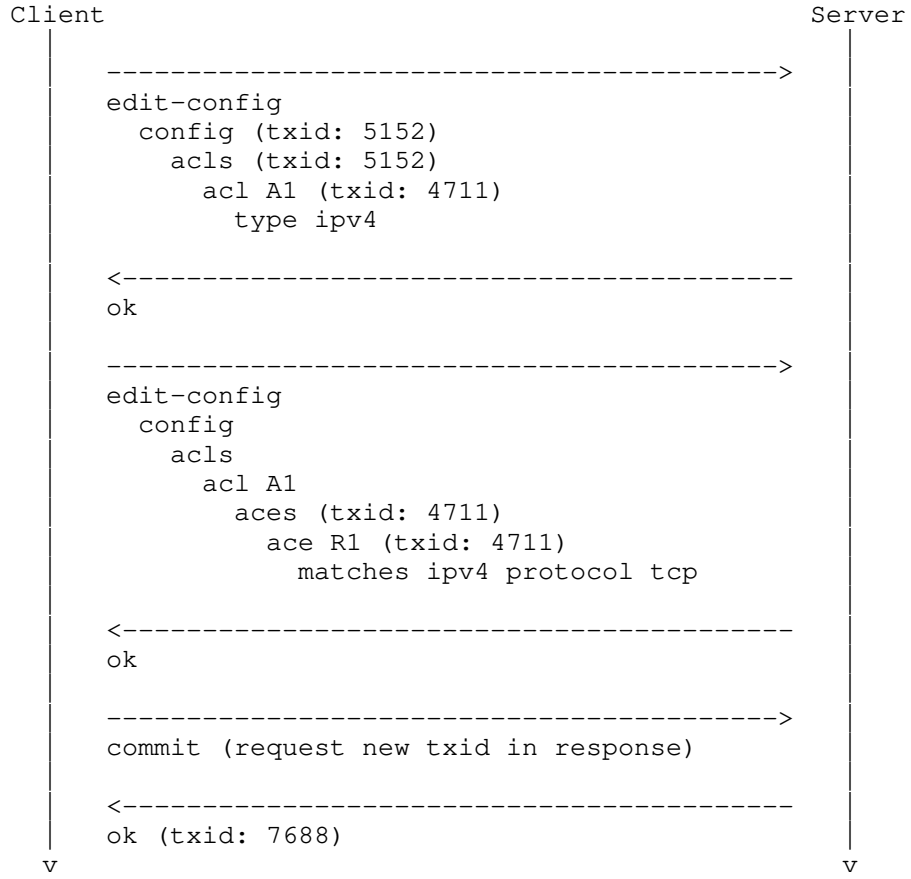

Figure 8: Conditional transaction towards the Candidate datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

3.6. Dependencies within Transactions

YANG modules that contain when-statements referencing remote parts of the model will cause the txid to change even in parts of the data tree that were not modified directly.

Let's say there is an energy-example.yang module that defines a mechanism for clients to request the server to measure the amount of energy that is consumed by a given access control rule. The energy-example module augments the access control module as follows:

```
  augment /acl:acls/acl:acl {
    when /energy-example:energy/energy-example:metering-enabled;
    leaf energy-tracing {
      type boolean;
      default false;
    }
    leaf energy-consumption {
      config false;
      type uint64;
      units J;
    }
  }
```

This means there is a system wide switch leaf metering-enabled in energy-example which disables all energy measurements in the system when set to false, and that there is a boolean leaf energy-tracing that controls whether energy measurement is happening for each acl rule individually.

In this example, we have an initial configuration like this:

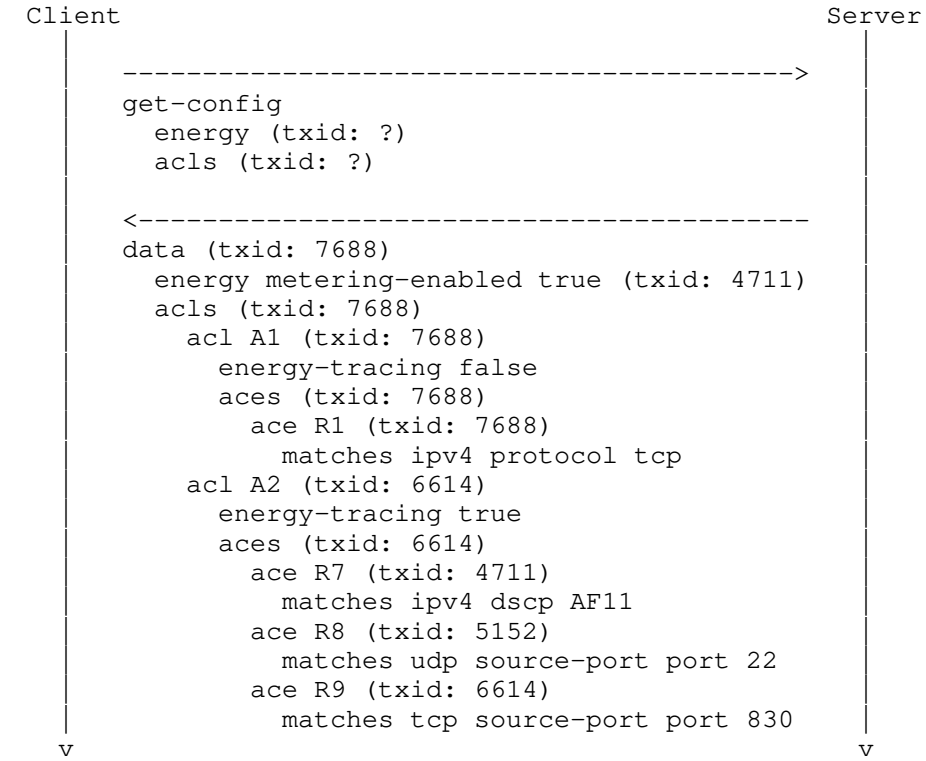

Figure 9: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

At this point, a client updates metering-enabled to false. This causes the when-expression on energy-tracing to turn false, removing the leaf entirely. This counts as a configuration change, and the txid MUST be updated appropriately.

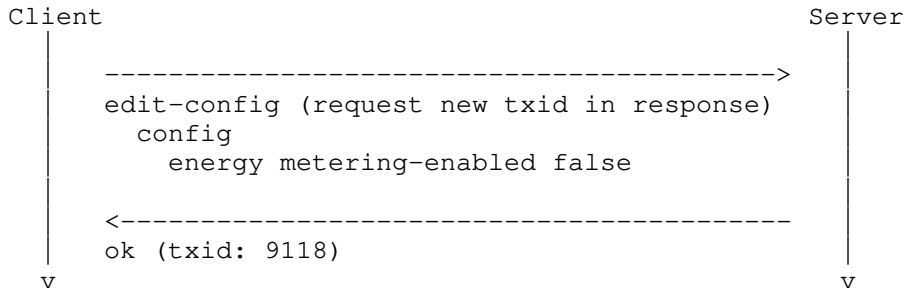

Figure 10: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in txid changes.

After the transaction above, the new configuration state has the energy-tracing leafs removed.

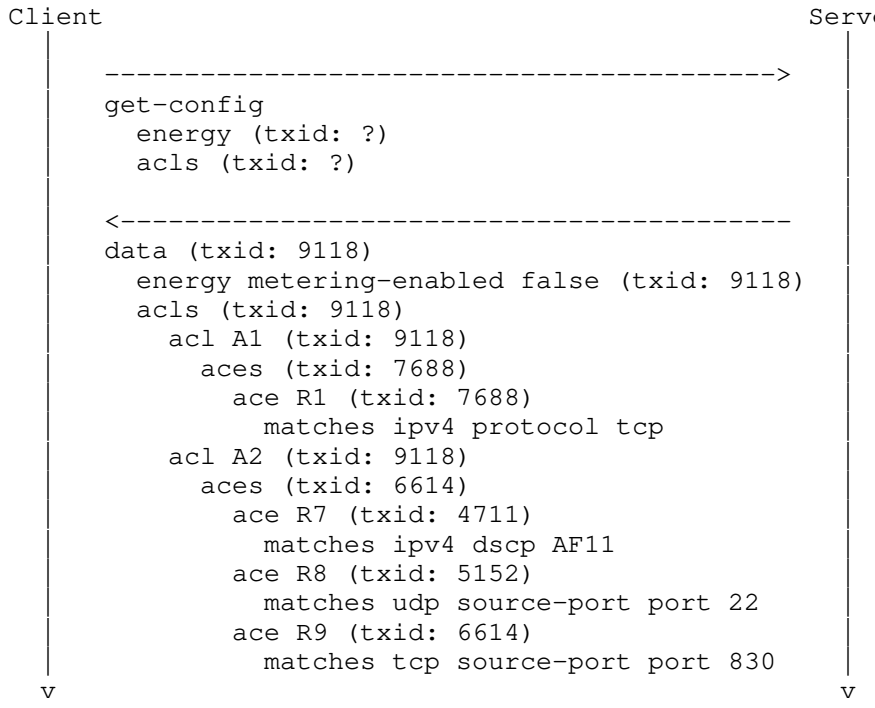

Figure 11: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when- expression.

3.7. Other NETCONF Operations

discard-changes The discard-changes operation resets the candidate datastore to the contents of the running datastore. The server MUST ensure the txid values in the candidate datastore get the same txid values as in the running datastore when this operation runs.

copy-config The copy-config operation can be used to copy contents between datastores. The server MUST ensure the txid values retain the same txid values as in the soruce datastore.

If copy-config is used to copy from a file, URL or other source that is not a datastore, the server MUST ensure the txid values are changed for the versioned nodes that are changed or have child nodes changed by the operation.

delete-config The server MUST ensure the datastore txid value is

changed, unless it was already empty.

commit At commit, with regards to the txid values, the server MUST treat the contents of the candidate datastore as if any txid value provided by the client when updating the candidate was provided in a single edit-config towards the running datastore. If the transaction is rejected due to txid value mismatch, an rpc-error as described in section Conditional Transactions (Section 3.5) MUST be sent.

3.8. YANG-Push Subscriptions

A client issuing a YANG-Push establish-subscription or modify-subscription request towards a server that supports both YANG-Push RFC 8641 (https://tools.ietf.org/html/rfc8641) and a txid mechanism MAY request that the server provides updated txid values in YANG-Push subscription updates.

4. Txid Mechanisms

This document defines two txid mechanisms:

* The etag attribute txid mechanism

* The last-modified attribute txid mechanism

Servers implementing this specification MUST support the etag attribute txid mechanism and MAY support the last-modified attribute txid mechanism.

Section NETCONF Txid Extension (Section 3) describes the logic that governs all txid mechanisms. This section describes the mapping from the generic logic to specific mechanism and encoding.

If a client uses more than one txid mechanism, such as both etag and last-modified in a particular message to a server, or patricular commit, the result is undefined.

4.1. The etag attribute txid mechanism

The etag txid mechanism described in this section is centered around a meta data XML attribute called "etag". The etag attribute is defined in the namespace "urn:ietf:params:xml:ns:netconf:txid:1.0". The etag attribute is added to XML elements in the NETCONF payload in order to indicate the txid value for the YANG node represented by the element.

NETCONF servers that support this extension MUST announce the capability "urn:ietf:params:netconf:capability:txid:etag:1.0".

The etag attribute values are opaque UTF-8 strings chosen freely, except that the etag string must not contain space, backslash or double quotes. The point of this restriction is to make it easy to reuse implementations that adhere to section 2.3.1 in RFC 7232 (https://tools.ietf.org/html/rfc7232). The probability SHOULD be made very low that an etag value that has been used historically by a server is used again by that server if the configuration is different.

It is RECOMMENDED that the same etag txid values are used across all management interfaces (i.e. NETCONF, RESTCONF and any other the server might implement), if it implements more than one.

The detailed rules for when to update the etag value are described in section General Txid Principles (Section 3.2). These rules are chosen to be consistent with the ETag mechanism in RESTCONF, RFC 8040 (https://tools.ietf.org/html/rfc8040), specifically sections 3.4.1.2, 3.4.1.3 and 3.5.2.

4.2. The last-modified attribute txid mechanism

The last-modified txid mechanism described in this section is centered around a meta data XML attribute called "last-modified". The last-modified attribute is defined in the namespace "urn:ietf:params:xml:ns:netconf:txid:1.0". The last-modified attribute is added to XML elements in the NETCONF payload in order to indicate the txid value for the YANG node represented by the element.

NETCONF servers that support this extension MUST announce the capability "urn:ietf:params:netconf:capability:txid:last-modified:1.0".

The last-modified attribute values are yang:date-and-time values as defined in ietf-yang-types.yang, RFC 6991 (https://datatracker.ietf.org/doc/html/rfc6991).

"2022-04-01T12:34:56.123456Z" is an example of what this time stamp format looks like. It is RECOMMENDED that the time stamps provided by the server to closely match the real world clock. Servers MUST ensure the timestamps provided are monotonously increasing for as long as the server’s operation is maintained.

It is RECOMMENDED that server implementors choose the number of digits of precision used for the fractional second timestamps high enough so that there is no risk that multiple transactions on the server would get the same timestamp.

It is RECOMMENDED that the same last-modified txid values are used across all management interfaces (i.e. NETCONF and any other the server might implement), except RESTCONF.

RESTCONF, as defined in RFC 8040 (https://tools.ietf.org/html/rfc8040), is using a different format for the time stamps which is limited to one second resolution. Server implementors that support the Last-Modified txid mechanism over both RESTCONF and other management protocols are RECOMMENDED to use Last-Modified timestamps that match the point in time referenced over RESTCONF, with the fractional seconds part added.

The detailed rules for when to update the last-modified value are described in section General Txid Principles (Section 3.2). These rules are chosen to be consistent with the Last-Modified mechanism in RESTCONF, RFC 8040 (https://tools.ietf.org/html/rfc8040), specifically sections 3.4.1.1, 3.4.1.3 and 3.5.1.

4.3. Common features to both etag and last-modified txid mechanisms

Clients MAY add etag or last-modified attributes to zero or more individual elements in the get-config or get-data filter, in which case they pertain to the subtree(s) rooted at the element(s) with the attributes.

Clients MAY also add such attributes directly to the get-config or get-data tags (e.g. if there is no filter), in which case it pertains to the txid value of the datastore root.

Clients might wish to send a txid value that is guaranteed to never match a server constructed txid. With both the etag and last-modified txid mechanisms, such a txid-request value is "?".

Clients MAY add etag or last-modified attributes to the payload of edit-config or edit-data requests, in which case they indicate the client's txid value of that element.

Clients MAY request servers that also implement YANG-Push to return configuration change subsription updates with etag or last-modified txid attributes. The client requests this service by adding a with-etag or with-last-modified flag with the value 'true' to the subscription request or yang-push configuration. The server MUST then return such txids on the YANG Patch edit tag and to the child

elements of the value tag. The txid attribute on the edit tag reflects the txid associated with the changes encoded in this edit section, as well as parent nodes. Later edit sections in the same push-update or push-change-update may still supercede the txid value for some or all of the nodes in the current edit section.

Servers returning txid values in get-config, edit-config, get-data, edit-data and commit operations MUST do so by adding etag and/or last-modified txid attributes to the data and ok tags. When servers prune output due to a matching txid value, the server MUST add a txid-match attribute to the pruned element, and MUST set the attribute value to "=", and MUST NOT send any element value.

Servers returning a txid mismatch error MUST return an rpc-error as defined in section Conditional Transactions (Section 3.5) with an error-info tag containing a txid-value-mismatch-error-info structure.

The txid attributes are valid on the following NETCONF tags, where xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0", xmlns:ncds="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda", xmlns:sn="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications", xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-patch" and xmlns:ypatch="urn:ietf:params:xml:ns:yang:ietf-yang-patch":

In client messages sent to a server:

* /nc:rpc/nc:get-config

* /nc:rpc/nc:get-config/nc:filter//*

* /nc:rpc/ncds:get-data

* /nc:rpc/ncds:get-data/ncds:subtree-filter//*

* /nc:rpc/ncds:get-data/ncds:xpath-filter//*

* /nc:rpc/nc:edit-config/nc:config

* /nc:rpc/nc:edit-config/nc:config//*

* /nc:rpc/ncds:edit-data/ncds:config

* /nc:rpc/ncds:edit-data/ncds:config//*

In server messages sent to a client:

* /nc:rpc-reply/nc:data

* /nc:rpc-reply/nc:data//*

* /nc:rpc-reply/ncds:data

* /nc:rpc-reply/ncds:data//*

* /nc:rpc-reply/nc:ok

* /yp:push-update/yp:datastore-contents/ypatch:yang-patch/
  ypatch:edit

* /yp:push-update/yp:datastore-contents/ypatch:yang-patch/
  ypatch:edit/ypatch:value//*

* /yp:push-change-update/yp:datastore-contents/ypatch:yang-patch/
  ypatch:edit

* /yp:push-change-update/yp:datastore-contents/ypatch:yang-patch/
  ypatch:edit/ypatch:value//*

5. Txid Mechanism Examples

5.1. Initial Configuration Response

5.1.1. With etag

NOTE: In the etag examples below, we have chosen to use a txid value consisting of "nc" followed by a monotonously increasing integer. This is convenient for the reader trying to make sense of the examples, but is not an implementation requirement. An etag would often be implemented as a "random" string of characters, with no comes-before/after relation defined.

To retrieve etag attributes across the entire NETCONF server configuration, a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config txid:etag="?"/>
</rpc>
```

The server’s reply might then be:

```
   <rpc-reply message-id="1"
              xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
              xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
     <data txid:etag="nc5152">
       <acls xmlns=
               "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
             txid:etag="nc5152">
         <acl txid:etag="nc4711">
           <name>A1</name>
           <aces txid:etag="nc4711">
             <ace txid:etag="nc4711">
               <name>R1</name>
               <matches>
                 <ipv4>
                   <protocol>udp</protocol>
                 </ipv4>
               </matches>
             </ace>
           </aces>
         </acl>
         <acl txid:etag="nc5152">
           <name>A2</name>
           <aces txid:etag="nc5152">
             <ace txid:etag="nc4711">
               <name>R7</name>
               <matches>
                 <ipv4>
                   <dscp>AF11</dscp>
                 </ipv4>
               </matches>
             </ace>
             <ace txid:etag="nc5152">
               <name>R8</name>
               <matches>
                 <udp>
                   <source-port>
                     <port>22</port>
                   </source-port>
                 </udp>
               </matches>
             </ace>
             <ace txid:etag="nc5152">
               <name>R9</name>
               <matches>
                 <tcp>
                   <source-port>
                     <port>22</port>
                   </source-port>
```

```
              </tcp>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
    <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"
          txid:etag="nc3072">
      <groups txid:etag="nc3072">
        <group txid:etag="nc3072">
          <name>admin</name>
          <user-name>sakura</user-name>
          <user-name>joe</user-name>
        </group>
      </groups>
    </nacm>
  </data>
</rpc>
```

To retrieve etag attributes for a specific ACL using an xpath filter, a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="xpath"
      xmlns:acl=
        "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      select="/acl:acls/acl:acl[acl:name='A1']"
      txid:etag="?"/>
  </get-config>
</rpc>
```

To retrieve etag attributes for "acls", but not for "nacm", a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="3"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:etag="?"/>
      <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
    </filter>
  </get-config>
</rpc>
```

If the server considers "acls", "acl", "aces" and "acl" to be versioned nodes, the server's response to the request above might look like:

```
<rpc-reply message-id="3"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls xmlns=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
          txid:etag="nc5152">
      <acl txid:etag="nc4711">
        <name>A1</name>
        <aces txid:etag="nc4711">
          <ace txid:etag="nc4711">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>udp</protocol>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
      <acl txid:etag="nc5152">
        <name>A2</name>
        <aces txid:etag="nc5152">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
```

```
              </matches>
            </ace>
            <ace txid:etag="nc5152">
              <name>R8</name>
              <matches>
                <udp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </udp>
              </matches>
            </ace>
            <ace txid:etag="nc5152">
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </tcp>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
      <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
        <groups>
          <group>
            <name>admin</name>
            <user-name>sakura</user-name>
            <user-name>joe</user-name>
          </group>
        </groups>
      </nacm>
    </data>
  </rpc>
```

5.1.2. With last-modified

To retrieve last-modified attributes for "acls", but not for "nacm", a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="4"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:last-modified="?"/>
      <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
    </filter>
  </get-config>
</rpc>
```

If the server considers "acls", "acl", "aces" and "acl" to be versioned nodes, the server's response to the request above might look like:

```
<rpc-reply message-id="4"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:last-modified="2022-04-01T12:34:56.789012Z">
      <acl txid:last-modified="2022-03-20T16:20:11.333444Z">
        <name>A1</name>
        <ace txid:last-modified="2022-03-20T16:20:11.333444Z">
          <name>R1</name>
          <matches>
            <ipv4>
              <protocol>udp</protocol>
            </ipv4>
          </matches>
        </ace>
      </acl>
      <acl txid:last-modified="2022-04-01T12:34:56.789012Z">
        <name>A2</name>
        <aces txid:last-modified="2022-04-01T12:34:56.789012Z">
          <ace txid:last-modified="2022-03-20T16:20:11.333444Z">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
```

```
            <ace txid:last-modified="2022-04-01T12:34:56.789012Z">
              <name>R8</name>
              <matches>
                <udp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </udp>
              </matches>
            </ace>
            <ace txid:last-modified="2022-04-01T12:34:56.789012Z">
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </tcp>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
      <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm"/>
        <groups>
          <group>
            <name>admin</name>
            <user-name>sakura</user-name>
            <user-name>joe</user-name>
          </group>
        </groups>
      </nacm>
    </data>
  </rpc>
```

5.2. Configuration Response Pruning

A NETCONF client that already knows some txid values MAY request that the configuration retrieval request is pruned with respect to the client's prior knowledge.

To retrieve only changes for "acls" that do not have the last known etag txid value, a client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="6"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:etag="nc5152">
        <acl txid:etag="nc4711">
          <name>A1</name>
          <aces txid:etag="nc4711"/>
        </acl>
        <acl txid:etag="nc5152">
          <name>A2</name>
          <aces txid:etag="nc5152"/>
        </acl>
    </filter>
  </get-config>
</rpc>
```

Assuming the NETCONF server configuration is the same as in the previous rpc-reply example, the server's response to request above might look like:

```
<rpc-reply message-id="6"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="="/>
  </data>
</rpc>
```

Or, if a configuration change has taken place under /acls since the client was last updated, the server's response may look like:

```
<rpc-reply message-id="6"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="nc6614">
      <acl txid:etag="=">
        <name>A1</name>
      </acl>
      <acl txid:etag="nc6614">
        <name>A2</name>
        <aces txid:etag="nc6614">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <ipv4>
                <source-port>
                  <port>22</port>
                </source-port>
              </ipv4>
            </matches>
          </ace>
          <ace txid:etag="nc6614">
            <name>R9</name>
            <matches>
              <ipv4>
                <source-port>
                  <port>830</port>
                </source-port>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc>
```

In case the client provides a txid value for a non-versioned node, the server needs to treat the node as having the same txid value as the closest ancestor that does have a txid value.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="7"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
        <acl>
          <name>A2</name>
          <aces>
            <ace>
              <name>R7</name>
              <matches>
                <ipv4>
                  <dscp txid:etag="nc4711"/>
                </ipv4>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
    </filter>
  </get-config>
</rpc>
```

If a txid value is specified for a leaf, and the txid value matches, the leaf value is pruned.

```
<rpc-reply message-id="7"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
      <acl>
        <name>A2</name>
        <aces>
          <ace>
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp txid:etag="="/>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

5.3. Configuration Change

A client that wishes to update the ace R1 protocol to tcp might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="8">
  <edit-config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
               xmlns:ietf-netconf-txid=
                "urn:ietf:params:xml:ns:yang:ietf-netconf-txid">
    <target>
      <running/>
    </target>
    <test-option>test-then-set</test-option>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
    <config>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:etag="nc5152">
        <acl txid:etag="nc4711">
          <name>A1</name>
          <aces txid:etag="nc4711">
            <ace txid:etag="nc4711">
              <matches>
                <ipv4>
                  <protocol>tcp</protocol>
                </ipv4>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
    </config>
  </edit-config>
</rpc>
```

The server would update the protocol leaf in the running datastore, and return an rpc-reply as follows:

```
<rpc-reply message-id="8"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <ok txid:etag="nc7688"/>
</rpc-reply>
```

A subsequent get-config request for "acls", with txid:etag="?" might then return:

```
<rpc-reply message-id="9"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
```

```
        txid:etag="nc7688">
        <acl txid:etag="nc7688">
          <name>A1</name>
          <aces txid:etag="nc7688">
            <ace txid:etag="nc7688">
              <name>R1</name>
              <matches>
                <ipv4>
                  <protocol>tcp</protocol>
                </ipv4>
              </matches>
            </ace>
          </aces>
        </acl>
        <acl txid:etag="nc6614">
          <name>A2</name>
          <aces txid:etag="nc6614">
            <ace txid:etag="nc4711">
              <name>R7</name>
              <matches>
                <ipv4>
                  <dscp>AF11</dscp>
                </ipv4>
              </matches>
            </ace>
            <ace txid:etag="nc5152">
              <name>R8</name>
              <matches>
                <udp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </udp>
              </matches>
            </ace>
            <ace txid:etag="nc6614">
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>830</port>
                  </source-port>
                </tcp>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
```

```
    </data>
  </rpc>
```

In case the server at this point received a configuration change from another source, such as a CLI operator, removing ace R8 and R9 in acl A2, a subsequent get-config request for acls, with txid:etag="?" might then return:

```
  <rpc-reply message-id="9"
             xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
             xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
    <data>
      <acls
        xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
        txid:etag="cli2222">
        <acl txid:etag="nc7688">
          <name>A1</name>
          <aces txid:etag="nc7688">
            <ace txid:etag="nc7688">
              <name>R1</name>
              <matches>
                <ipv4>
                  <protocol>tcp</protocol>
                </ipv4>
              </matches>
            </ace>
          </aces>
        </acl>
        <acl txid:etag="cli2222">
          <name>A2</name>
          <aces txid:etag="cli2222">
            <ace txid:etag="nc4711">
              <name>R7</name>
              <matches>
                <ipv4>
                  <dscp>AF11</dscp>
                </ipv4>
              </matches>
            </ace>
          </aces>
        </acl>
      </acls>
    </data>
  </rpc>
```

5.4. Conditional Configuration Change

If a client wishes to delete acl A1 if and only if its configuration has not been altered since this client last synchronized its configuration with the server, at which point it received the etag "nc7688" for acl A1, regardless of any possible changes to other acls, it might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="10"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0"
     xmlns:ietf-netconf-txid=
       "urn:ietf:params:xml:ns:yang:ietf-netconf-txid">
  <edit-config>
    <target>
      <runnign/>
    </target>
    <test-option>test-then-set</test-option>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
    <config>
      <acls xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
        <acl nc:operation="delete"
             txid:etag="nc7688">
          <name>A1</name>
        </acl>
      </acls>
    </config>
  </edit-config>
</rpc>
```

If acl A1 now has the etag txid value "nc7688", as expected by the client, the transaction goes through, and the server responds something like:

```
<rpc-reply message-id="10"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <ok txid:etag="nc8008"/>
</rpc-reply>
```

A subsequent get-config request for acls, with txid:etag="?" might then return:

```
<rpc-reply message-id="11"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="nc8008">
      <acl txid:etag="cli2222">
        <name>A2</name>
        <aces txid:etag="cli2222">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>AF11</dscp>
              </ipv4>
            </matches>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc>
```

In case acl A1 did not have the expected etag txid value "nc7688", when the server processed this request, it rejects the transaction, and might send:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:acl=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
           xmlns:ietf-netconf-txid=
             "urn:ietf:params:xml:ns:yang:ietf-netconf-txid"
           message-id="11">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-info>
      <ietf-netconf-txid:txid-value-mismatch-error-info>
        <ietf-netconf-txid:mismatch-path>
          /acl:acls/acl:acl[acl:name="A1"]
        </ietf-netconf-txid:mismatch-path>
        <ietf-netconf-txid:mismatch-etag-value>
          cli6912
        </ietf-netconf-txid:mismatch-etag-value>
      </ietf-netconf-txid:txid-value-mismatch-error-info>
    </error-info>
  </rpc-error>
</rpc-reply>
```

5.5. Using etags with Other NETCONF Operations

The client MAY request that the new etag txid value is returned as an attribute on the ok response for a successful commit. The client requests this by adding with-etag to the commit operation.

For example, a client might send:

```
<rpc message-id="12"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    xmlns:ietf-netconf-txid=
      "urn:ietf:params:xml:ns:yang:ietf-netconf-txid"
  <commit>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
  </commit>
</rpc>
```

Assuming the server accepted the transaction, it might respond:

```
<rpc-reply message-id="12"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <ok txid:etag="nc8008"/>
</rpc-reply>
```

5.6. YANG-Push

A client MAY request that the updates for one or more YANG Push subscriptions are annotated with the txid values. The request might look like this:

```
<netconf:rpc message-id="13"
             xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
      xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
      xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push"
      xmlns:ietf-netconf-txid-yp=
        "urn:ietf:params:xml:ns:yang:ietf-txid-yang-push">
    <yp:datastore
        xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:running
    </yp:datastore>
    <yp:datastore-xpath-filter
        xmlns:acl=
          "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
      /acl:acls
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
    <ietf-netconf-txid-yp:with-etag>
      true
    </ietf-netconf-txid-yp:with-etag>
  </establish-subscription>
</netconf:rpc>
```

In case a client wishes to modify a previous subscription request in order to no longer receive YANG Push subscription updates, the request might look like this:

```
<rpc message-id="14"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
      xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
      xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push"
      xmlns:ietf-netconf-txid-yp=
        "urn:ietf:params:xml:ns:yang:ietf-txid-yang-push">
    <id>1011</id>
    <yp:datastore
        xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:running
    </yp:datastore>
    <ietf-netconf-txid-yp:with-etag>
      false
    </ietf-netconf-txid-yp:with-etag>
  </modify-subscription>
</rpc>
```

A server might send a subscription update like this:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2022-04-04T06:00:24.16Z</eventTime>
  <push-change-update
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>89</id>
    <datastore-changes>
      <yang-patch>
        <patch-id>0</patch-id>
        <edit txid:etag="nc8008">
          <edit-id>edit1</edit-id>
          <operation>delete</operation>
          <target xmlns:acl=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
            /acl:acls
          </target>
          <value>
            <acl xmlns=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
              <name>A1</name>
            </acl>
          </value>
        </edit>
      </yang-patch>
    </datastore-changes>
  </push-change-update>
</notification>
```

6. YANG Modules

6.1. Base module for txid in NETCONF

```
   module ietf-netconf-txid {
     yang-version 1.1;
     namespace
       'urn:ietf:params:xml:ns:yang:ietf-netconf-txid';
     prefix ietf-netconf-txid;

     import ietf-netconf {
       prefix nc;
     }

     import ietf-netconf-nmda {
       prefix ncds;
     }

     import ietf-yang-structure-ext {
       prefix sx;
     }

     import ietf-yang-types {
       prefix yang;
     }

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

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

        Author:   Jan Lindblad
                  <mailto:jlindbla@cisco.com>";

     description
       "NETCONF Transaction ID aware operations for NMDA.

        Copyright (c) 2022 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

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

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

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

    revision 2022-04-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Xxxxxxxxx";
    }

    typedef etag-t {
      type string {
        pattern ".* .*" {
          modifier invert-match;
        }
        pattern '.*".*' {
          modifier invert-match;
        }
        pattern ".*\\.*" {
          modifier invert-match;
        }
      }
      description
        "Unique Entity-tag txid value representing a specific
        transaction.  Could be any string that does not contain
        spaces, double quotes or backslash.  The txid values '?'
        and '=' have special meaning.";
    }

    typedef last-modified-t {
      type union {
        type yang:date-and-time;
        type enumeration {
          enum ? {
            description "Txid value used by clients that is
              guaranteed not to match any txid on the server.";
          }
          enum = {
            description "Txid value used by servers to indicate
              that contents has been pruned due to txid match
```

```
              between client and server.";
          }
        }
      }
      description
        "Last-modified txid value representing a specific transaction.
         The txid values '?' and '=' have special meaning.";
    }

    grouping txid-grouping {
      leaf with-etag {
        type boolean;
        description
          "Indicates whether the client requests the server to include
           a txid:etag txid attribute when the configuration has
           changed.";
      }
      leaf with-last-modified {
        type boolean;
        description
          "Indicates whether the client requests the server to include
           a txid:last-modified attribute when the configuration has
           changed.";
      }
      description
        "Grouping for txid mechanisms, to be augmented into
         rpcs that modify configuration data stores.";
    }

    augment /nc:edit-config/nc:input {
      uses txid-grouping;
      description
        "Injects the txid mechanisms into the
        edit-config operation";
    }

    augment /nc:commit/nc:input {
      uses txid-grouping;
      description
        "Injects the txid mechanisms into the
        commit operation";
    }

    augment /ncds:edit-data/ncds:input {
      uses txid-grouping;
      description
        "Injects the txid mechanisms into the
        edit-data operation";
```

```
    }

    sx:structure txid-value-mismatch-error-info {
      container txid-value-mismatch-error-info {
        description
           "This error is returned by a NETCONF server when a client
            sends a configuration change request, with the additonal
            condition that the server aborts the transaction if the
            server's configuration has changed from what the client
            expects, and the configuration is found not to actually
            not match the client's expectation.";
        leaf mismatch-path {
          type instance-identifier;
          description
            "Indicates the YANG path to the element with a mismatching
             etag txid value.";
        }
        leaf mismatch-etag-value {
          type etag-t;
          description
            "Indicates server's txid value of the etag
            attribute for one mismatching element.";
        }
        leaf mismatch-last-modified-value {
          type last-modified-t;
          description
            "Indicates server's txid value of the last-modified
            attribute for one mismatching element.";
        }
      }
    }
  }
```

6.2. Additional support for txid in YANG-Push

```
  module ietf-netconf-txid-yang-push {
    yang-version 1.1;
    namespace
      'urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push';
    prefix ietf-netconf-txid-yp;

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

    import ietf-netconf-txid {
```

```
    prefix ietf-netconf-txid;
    reference
      "RFC XXXX: Xxxxxxxxx";
  }

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

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

     Author:   Jan Lindblad
               <mailto:jlindbla@cisco.com>";

  description
    "NETCONF Transaction ID aware operations for YANG Push.

     Copyright (c) 2022 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

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

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

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

  revision 2022-04-01 {
    description
      "Initial revision";
    reference
      "RFC XXXX: Xxxxxxxxx";
  }

  augment "/sn:establish-subscription/sn:input" {
    description
```

```
        "This augmentation adds additional subscription parameters
          that apply specifically to datastore updates to RPC input.";
      uses ietf-netconf-txid:txid-grouping;
    }
    augment "/sn:modify-subscription/sn:input" {
      description
        "This augmentation adds additional subscription parameters
          specific to datastore updates.";
      uses ietf-netconf-txid:txid-grouping;
    }
    augment "/sn:subscriptions/sn:subscription" {
      description
        "This augmentation adds additional subscription parameters
          specific to datastore updates.";
      uses ietf-netconf-txid:txid-grouping;
    }
  }
```

7. Security Considerations

TODO Security

8. IANA Considerations

This document registers the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry:

  urn:ietf:params:netconf:capability:txid:1.0

This document registers three XML namespace URNs in the 'IETF XML registry', following the format defined in RFC 3688 (https://tools.ietf.org/html/rfc3688).

  URI: urn:ietf:params:xml:ns:netconf:txid:1.0

  URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid

  URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push

  Registrant Contact: The NETCONF WG of the IETF.

  XML: N/A, the requested URIs are XML namespaces.

This document registers two module names in the 'YANG Module Names' registry, defined in RFC 6020 (https://tools.ietf.org/html/rfc6020).

name: ietf-netconf-txid

prefix: ietf-netconf-txid

namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid

RFC: XXXX

and

name: ietf-netconf-txid-yp

prefix: ietf-netconf-txid-yp

namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push

RFC: XXXX

9. Changes

9.1. Major changes in -02 since -01

* A last-modified txid mechanism has been added (back). This mechanism aligns well with the Last-Modified mechanism defined in RESTCONF RFC 8040 (https://tools.ietf.org/html/rfc8040), but is not a carbon copy.

* YANG Push functionality has been added. This allows YANG Push users to receive txid updates as part of the configuration updates. This functionality comes in a separate YANG module, to allow implementors to cleanly keep all this functionality out.

* Changed name of "versioned elements". They are now called "versioned nodes".

* Clarified txid behavior for transactions toward the Candidate datastore, and some not so common situations, such as when a client specifies a txid for a non-versioned node, and when there are when-statement dependencies across subtrees.

* Examples provided for the abstract mechanism level with simple message flow diagrams.

* More examples on protocol level, and with ietf-interfaces as example target module replaced with ietf-access-control to reduce confusion.

* Explicit list of XPaths to clearly state where etag or last-modified attributes may be added by clients and servers.

* Document introduction restructured to remove duplication between sections and to allow multiple (etag and last-modified) txid mechanisms.

* Moved the actual YANG module code into proper module files that are included in the source document. These modules can be compiled as proper modules without any extraction tools.

9.2. Major changes in -01 since -00

* Updated the text on numerous points in order to answer questions that appeared on the mailing list.

* Changed the document structure into a general transaction id part and one etag specific part.

* Renamed entag attribute to etag, prefix to txid, namespace to urn:ietf:params:xml:ns:yang:ietf-netconf-txid.

* Set capability string to urn:ietf:params:netconf:capability:txid:1.0

* Changed YANG module name, namespace and prefix to match names above.

* Harmonized/slightly adjusted etag value space with RFC 7232 and RFC 8040.

* Removed all text discussing etag values provided by the client (although this is still an interesting idea, if you ask the author)

* Clarified the etag attribute mechanism, especially when it comes to matching against non-versioned elements, its cascading upwards in the tree and secondary effects from when- and choice-statements.

* Added a mechanism for returning the server assigned etag value in get-config and get-data.

* Added section describing how the NETCONF discard-changes, copy-config, delete-config and commit operations work with respect to etags.

* Added IANA Considerations section.

* Removed all comments about open questions.

10. Normative References

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

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

Acknowledgments

The author wishes to thank Benoit Claise for making this work happen, and the following individuals, who all provided helpful comments: Per Andersson, Kent Watsen, Andy Bierman, Robert Wilton, Qiufang Ma.

Author's Address

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com

Network Configuration
Internet-Draft
Intended status: Standards Track
Expires: 19 December 2022

S. Turner
sn3rd
R. Housley
Vigil Security
17 June 2022

NETCONF over TLS 1.3
draft-turner-netconf-over-tls13-00

Abstract

RFC 7589 defines how to protect NETCONF messages with TLS 1.2. This document describes how to protect NETCONF messages with TLS 1.3.

Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://github.com/seanturner/netconf-over-tls13.

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 19 December 2022.

Copyright Notice

Table of Contents

1. Introduction

[RFC7589] defines how to protect NETCONF messages [RFC6241] with TLS 1.2 [RFC5246]. This document describes defines how to protect NETCONF messages with TLS 1.3 [I-D.ietf-tls-rfc8446bis].

This document addresses cipher suites and the use of early data, which is also known as 0-RTT data. It also updates the "netconf-tls" IANA Registered Port Number entry to refer to this document. All other provisions set forth in [RFC7589] are unchanged, including connection initiation, message framing, connection closure, certificate validation, server identity, and client identity.

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. Early Data

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. Early data is permitted by TLS 1.3 when the client and server share a 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 are no protection against the replay of early data between connections. Appendix E.5 of [I-D.ietf-tls-rfc8446bis] requires applicaitons not use early data without a profile that defines its use. This document specifies that implementations MUST NOT use early data.

4. Cipher Suites

Implementations MUST support TLS 1.3 [I-D.ietf-tls-rfc8446bis], and implementation are REQUIRED to support the mandatory-to-implement cipher suites listed in Section 9.1 of [I-D.ietf-tls-rfc8446bis].

Implementations MAY implement additional TLS cipher suites that provide mutual authentication and confidentiality, which are required for NETCONF [RFC6241].

Implementations SHOULD follow the recommendations given in [I-D.ietf-uta-rfc7525bis].

So, this is what {{Section 9.1 of I-D.ietf-tls-rfc8446bis}} says:

A TLS-compliant application MUST implement the TLS_AES_128_GCM_SHA256 [GCM] cipher suite and SHOULD implement the TLS_AES_256_GCM_SHA384 [GCM] and TLS_CHACHA20_POLY1305_SHA256 [RFC8439] cipher suites (see Appendix B.4).

A TLS-compliant application MUST support digital signatures with rsa_pkcs1_sha256 (for certificates), rsa_pss_rsae_sha256 (for CertificateVerify and certificates), and ecdsa_secp256r1_sha256. A TLS-compliant application MUST support key exchange with secp256r1 (NIST P-256) and SHOULD support key exchange with X25519 [RFC7748].

Is there any reason to narrow the algorithm choices?

My guess is not. These ought to be available in all TLS libraries.

5. Security Considerations

Please review the Security Considerations in TLS 1.3 [I-D.ietf-tls-rfc8446bis].

Please review the recommendations regarding Diffie-Hellman exponent reuse in Section 7.4 of [I-D.ietf-uta-rfc7525bis].

Please review the Security Considerations in NETCONF [RFC6241].

NETCONF is used to access configuration and state information and to modify configuration information. TLS 1.3 mutual authentication is used to ensure that only authorized users and systems are able to view the NETCONF server's configuration and state or to modify the NETCONF server's configuration. To this end, neither the client nor the server should establish a NETCONF over TLS 1.3 connection with an unknown, unexpected, or incorrect peer identity; see Section 7 of [RFC7589]. If deployments make use of this list of Certification Authority (CA) certificates [RFC5280], then the listed CAs should only issue certificates to parties that are authorized to access the NETCONF servers. Doing otherwise will allow certificates that were issued for other purposes to be inappropriately accepted by a NETCONF server.

Please review [RFC6125] for further details on generic host name validation in the TLS context.

Please review the recommendations regarding certificate revocation checking in Section 7.5 of [I-D.ietf-uta-rfc7525bis].

[RFC5539] assumes that the end-of-message (EOM) sequence,]]>]]>, cannot appear in any well-formed XML document, which turned out to be mistaken. The EOM sequence can cause operational problems and open space for attacks if sent deliberately in NETCONF messages. While it is possible, the likelihood is believed to be very low. The EOM sequence is used for the initial <hello> message to avoid incompatibility with existing implementations. When the client and server both implement the :base:1.1 capability, a proper framing protocol (see Section 3 of [RFC7589]) is used for the rest of the NETCONF session, to avoid injection attacks.

6. IANA Considerations

IANA is requested to add a reference to this document in the "netconf-tls" entry in the "Registered Port Numbers". The updated registry entry would appear as follows:

Service Name:	netconf-tls
Transport Protocol(s):	TCP
Assignee:	IESG <iesg@ietf.org>
Contact:	IETF Chair <chair@ietf.org>
Description:	NETCONF over TLS
Reference:	RFC 7589, [THIS RFC]
Port Number:	6513

7. References

7.1. Normative References

[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-04, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-04>.

[I-D.ietf-uta-rfc7525bis]
Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", Work in Progress, Internet-Draft, draft-ietf-uta-rfc7525bis-07, 26 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-uta-rfc7525bis-07>.

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC5539] Badra, M., "NETCONF over Transport Layer Security (TLS)", RFC 5539, DOI 10.17487/RFC5539, May 2009, <https://www.rfc-editor.org/rfc/rfc5539>.

[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>.

[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/rfc/rfc7589>.

[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>.

7.2. Informative References

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/rfc/rfc5246>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/rfc/rfc6125>.

Acknowledgments

We would like to thank the following people TBD.

Authors' Addresses

Sean Turner
sn3rd
Email: sean@sn3rd.com

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA, 20170
United States of America
Email: housley@vigilsec.com