NETCONF Working Group
Internet-Draft
Intended status: Experimental
Expires: 24 April 2023

Q. Wu
W. Song
Huawei
P. Liu
China Mobile
Q. Ma
Huawei
W. Wang
China Telecom
Z. Niu
Microsoft

21 October 2022

Adaptive Subscription to YANG Notification
draft-ietf-netconf-adaptive-subscription-01

Abstract

This document defines a YANG data model and associated mechanism enabling the subscriber's adaptive subscriptions to a publisher's event streams with various different period intervals to report updates. Applying these elements allows servers automatically adjust the rate and volume of telemetry traffic sent from a publisher to receivers.

Status of This Memo

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

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

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

This Internet-Draft will expire on 24 April 2023.

Table of Contents

1. Introduction

YANG-Push subscriptions [RFC8641] allow subscriber applications to request a continuous customized stream of updates from a YANG datastore without needing to poll. It defines a mechanism (i.e., update trigger) to determine when an update record needs to be generated. Two types of subscriptions are introduced in [RFC8641], distinguished by how updates are triggered: periodic and on-change.

* Periodic subscription allows subscribed data to be streamed to the destination at a configured fixed periodic interval;

* On-change subscription allows update to be triggered whenever a change in the subscribed information is detected.

However in some large scale deployments (e.g., massive data collection for wireless network performance monitoring) where an increased data collection rate is used, it becomes more likely that both clients and servers are temporarily overwhelmed with a burst of streamed data and consumes expensive network resource (e.g., bandwidth resource, radio resource) and computation resource, therefore hard to continuously monitor operational data, especially values that fall outside normal operational ranges. If the rate at which we can collect a stream of data is set too low or chosen to get low priority telemetry data dropped, these telemetry data are not sufficient to detect and diagnose problems and verify correct network behavior.

A client might choose to monitor the operational state and send a request to modify the data collection rate on the server. But how often the client evaluates if the modification of the data collection rate is required highly depends on the current collection rate, collecting a stream of data at a low rate prevents the subscriber from capturing sufficient data for timely decision-making, which may result in service discontinuity. In addition, when tens of thousands of network devices need to be managed, frequent follow-up modification requests are prone to errors.

There is a need for a service to balance between data management cost and real time streaming telemetry. To achieve this, servers can be configured with multiple different period intervals and corresponding subscription update policy which allows servers/publishers automatically switch to different period intervals according to the network condition change without the interaction with the client for policy update instruction, e.g., when the wireless signal strength falls below a configured threshold, the subscribed data can be streamed at a higher rate to capture potentially important data and events (e.g., continuous service degeneration); while when the wireless signal strength crosses a configured threshold, the subscribed data can be streamed at a lower rate.

This document defines a YANG data model and associated mechanism enabling the subscriber’s adaptive subscriptions to a publisher’s event streams. Applying these elements allows servers to automatically adjust the rate and volume of telemetry traffic sent from a publisher to receivers.

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 [RFC5277] [RFC7950] [RFC3198] [RFC8342] [RFC8639] [RFC8641] and are not redefined here:

* Event
* Client
* Configuration
* Configured subscription
* Configuration datastore
* Notification message
* Publisher
* Receiver
* Subscriber
* Subscription
* On-change subscription
* Periodic subscription
* Selection filter

This document defines the following term:

Adaptive Subscription: Apply subscription update policy on the servers and allow servers/publishers automatically switch to different period intervals according to the network condition change without the interaction with the client for update policy instruction.

2. Model Overview

This document defines a YANG module "ietf-adaptive-subscription", which augments the "update-trigger" choice defined in the "ietf-yang-push" module [RFC8641] with subscription configuration parameters that are specific to a subscriber's adaptive subscription.

In addition to subscription state notifications defined in [RFC8639] and notifications for subscribed content defined in [RFC8641], "ietf-adaptive-subscription" YANG module also defines "adaptive-period-update" notification to report the update interval change.

The following tree diagrams [RFC8340] provide an overview of the data model for "ietf-adaptive-subscription" module.

```
module: ietf-adaptive-subscription
  augment /sn:subscriptions/sn:subscription/yp:update-trigger:
    +--:(adaptive-subscriptions)
       +--rw adaptive-subscriptions
          +--rw adaptive-period* [name]
             +--rw name                    string
             +--rw xpath-external-eval     string
             +--rw period                  yp:centiseconds
             +--rw anchor-time?            yang:date-and-time
  augment /sn:establish-subscription/sn:input/yp:update-trigger:
    +--:(adaptive-subscriptions)
       +-- adaptive-subscriptions
          +-- adaptive-period* [name]
             +-- name                    string
             +-- xpath-external-eval     string
             +-- period                  yp:centiseconds
             +-- anchor-time?            yang:date-and-time
  notifications:
    +---n adaptive-period-update
       +--ro id?                                    sn:subscription-id
       +--ro period                                 yp:centiseconds
       +--ro anchor-time?                           yang:date-and-time
       +--ro datastore                              identityref
       +--ro (selection-filter)?
          +--:(by-reference)
          |  +--ro selection-filter-ref             selection-filter-ref
          +--:(within-subscription)
             +--ro (filter-spec)?
                +--:(datastore-subtree-filter)
                |  +--ro datastore-subtree-filter?   <anydata> {sn:subtree}?
                +--:(datastore-xpath-filter)
                   +--ro datastore-xpath-filter?     yang:xpath1.0 {sn:xpath}?
```

2.1. Subscription Configuration

For adaptive subscriptions, triggered updates will occur at the boundaries of specified time intervals when a trigger condition is satisfied. These boundaries can be calculated from the following adaptive periodic parameters:

* a "name" represents the name of each adaptive period;

* a "period" defines the new duration between push updates. The period can be switched based on trigger conditions;

* an "anchor-time"; update intervals fall on the points in time that are a multiple of a "period" from an "anchor-time". If an "anchor-time" is not provided, then the "anchor-time" MUST be set with the creation time of the initial update record.

* an "xpath-external-eval" represents a standard XPath evaluation expression (See section 6.4 of [RFC7950]) that is applied against the targeted data object, which is used to trigger/control the update interval switching within the server. It follows the rules defined in section 3.4 of [XPATH1.0] and contains comparisons of the targeted datastore node with its value to the specific threshold in the XPath format. Different from selection filter defined in [RFC8641],

  - it is applied against a single targeted object rather than a set of target objects.

  - it monitors a specific data object change and evaluates the trigger condition associated with the targeted object to be true or false using XPATH rules and does not influence the event records output generation from a publisher.

  How often the XPath expression criterion is evaluated is up to the publisher's implementation. With minimal delay, the expression can be evaluated whenever changes to targeted data object occur, or at the end of each higher frequency streaming update period. To reduce the frequency of evaluation, the server can choose to check targeted object change at every multiple (e.g.,2 or 3) high-frequency streaming update periods.

  The represented expression defined in "xpath-external-eval" is evaluated in the following XPath context:

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

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

- The set of variable bindings is empty.

- The function library is the core function library defined in [XPATH1.0] and the function defined in Section 10 in RFC 7950.

- The context node is the root node.

For the cases where multiple list instances are needed to handle in "xpath-external-eval", XPath abbreviated syntax can be used to identify a particular instance, e.g., to represent a comparison for a leaf in a list entry:

```
/if:interfaces/if:interface[if:name="eth0"]/if:in-errors>1000.
```

The server MUST convert the XPath expression defined in "xpath-external-eval" to a boolean value and internally apply the "boolean" function defined in Section 4.3 in [XPATH1.0] if the evaluated result is not a boolean value.

Note that the adaptive subscription may not be supported by every YANG datastore node. A publisher MAY decide to simply reject an adaptive subscription with "adaptive-unsupported" (defined in Section 2.2.1.1) if the scope of the subscription contains selected data nodes for which adaptive subscription is not supported.

2.2. YANG RPC

2.2.1. "establish-subscription" RPC

The augmentation of YANG module "ietf-yang-push" made to RPCs specified in YANG module "ietf-subscribed-notifications" [RFC8639] is introduced. This augmentation concerns the "establish-subscription" RPC, which is augmented with parameters that are needed to specify a subscriber's adaptive subscriptions. These parameters are the same as ones defined in Section 2.1.

2.2.1.1. RPC Failures

As specified in [RFC8639] and [RFC8641], RPC error responses from the publisher are used to indicate a rejection of an RPC for any reason. This document introduces three new RPC errors for "establish-subscription" RPC.

```
establish-subscription
------------------------------
adaptive-unsupported
xpath-evaluation-unsupported
multi-xpath-criteria-conflict
```

Adaptive-unsupported is used to indicate that the adaptive subscription is not supported for any objects that are selectable by the filter.

Xpath-evaluation-unsupported is used to indicate that a server fails to parse syntax defined in "xpath-external-eval". The failure can be caused by either a syntax error or some XPath 1.0 syntax not supported against the specific object.

Multi-xpath-criteria-conflict is used to indicate that the multiple Xpath evaluation criteria represented by "xpath-external-eval" is evaluated as conflict, i.e., more than one condition expressions are evaluated to "true". However, the publisher should still push updates at the higher frequency streaming period if multiple Xpath evaluations conflict with each other during the lifecycle of an adaptive subscription.

For an example of how above RPC errors can be returned, see the "xpath-evaluation-unsupported" error response illustrated in Appendix B.3.

Note that existing RPC errors defined in RFC 8639 and RFC 8641 are still supported by this document. For example, if any configured period for adaptive subscription is not supported by the publisher, a "period-unsupported" error response could be used.

2.3. Notifications for Adaptive Subscribed Content

The adaptive update notification is similar to subscription state change notifications defined in [RFC8639]. It is inserted into the sequence of notification messages sent to a particular receiver. As stated in RFC 8639, section 2.7, the adaptive update notification cannot be dropped or filtered out, it cannot be stored in replay buffers, and it is delivered only to impacted receivers of a subscription. The identification of adaptive update notification is

easy to separate from other notification messages through the use of the YANG extension "subscription-state-notif". This extension tags a notification as a subscription state change notification.

The objects in the 'adaptive-period-update' notification include:

* a "period" that defines the duration between push updates, the period can be changed based on trigger conditions.

* an "anchor-time"; update intervals fall on the points in time that are a multiple of a "period" from an "anchor-time". If an "anchor-time" is not provided, then the "anchor-time" MUST be set with the creation time of the initial update record.

* A selection filter is to identify YANG nodes of interest in a datastore. Filter contents are specified via a reference to an existing filter or via an in-line definition for only that subscription based on XPath Evaluation criteria defined in section 6.4 of [RFC7950]. Referenced filters allow an implementation to avoid evaluating filter acceptability during a dynamic subscription request. The "case" statement differentiates the options. Note that filter contents are not affected by "xpath-external-eval" parameter defined by the update trigger.

3. XPath Complexity Evaluation

YANG-Push subscriptions [RFC8641] specify selection filters to identify targeted YANG datastore nodes and/or datastore subtrees for which updates are to be pushed. In addition, it specifies update policies which contain conditions that trigger generation and pushing of new update records. To support a subscriber's adaptive subscription defined in this document, the trigger condition can also use similar selection filters to express a standard XPath Evaluation criterion (section 6.4 of [RFC7950]) against targeted data objects.

Similar to on-change subscriptions, adaptive subscriptions are particularly effective for data that changes infrequently, the following complex design choices need to be cautious, although these designs have already been well supported by the section 3.4 of [XPATH1.0]:

* Support XPath Evaluation criteria against every data object;

* Support more than one target object selection and operation(e.g., addition, subtraction, division and multiplication) in the XPath evaluation criterion;

* Support any type of node set in the XPath evaluation criterion, e.g., string,int64, uint64, and decimal64 types;

* Both objects in the XPath Evaluation criterion to be compared are node-sets;

* Two objects to be compared are in different data types, e.g., one is an integer, the other is a string

As described in section 6.4 of RFC7950, Numbers in XPath 1.0 are IEEE 754 [IEEE754-2008] double-precision floating-point values; some values of int64, uint64, and decimal64 types cannot be exactly represented in XPath expressions.

If two objects to be compared are in different data types, conversion function is needed to convert different data types into numbers.

If both objects in XPath Evaluation criteria to be compared are node-sets, more computation resources are required which add complexity.

To reduce these complexities, the following design principles are recommended:

* XPath Evaluation criteria against a minimal set of data objects in the data model, the minimal set of data objects can be advertised using Notification capabilities model defined in [RFC9196].

* XPath Evaluation criteria only support condition expressions that filter updates based on numbers.

* One object to be compared in the XPath Evaluation criteria MUST be a leaf data node.

* The other object to be compared in the XPath Evaluation criteria MUST be a number data type.

If a server receives an XPath Evaluation criterion with some XPath syntax unsupported against the specific object, an RPC error with "xpath-evaluation-unsupported" should be returned.

4. Adaptive Subscription YANG Module

<CODE BEGINS> file "ietf-adaptive-subscription@2022-10-21.yang"

```
module ietf-adaptive-subscription {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-adaptive-subscription";
  prefix as;
```

```
  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-yang-push {
    prefix yp;
    reference
      "RFC 8641: Subscription to YANG Notifications for Datastore Updates";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

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

     Editor: Qin Wu
           <mailto:bill.wu@huawei.com>

     Editor: Wei Song
           <mailto: songwei80@huawei.com>

     Editor: Peng Liu
           <mailto: liupengyjy@chinamobile.com>

     Editor: Qiufang Ma
           <mailto: maqiufang1@huawei.com>

     Editor: Wei Wang
          <mailto: wangw36@chinatelecom.cn>

     Editor: Zhixiong Niu
          <mailto: Zhixiong.Niu@microsoft.com>";
  description
    "This module extends the YANG data module defined in
     YANG-push to enable the subscriber's adaptive
     subscriptions to a publisher's event streams with various
     different period intervals to report updates.

     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 2022-10-21 {
      description
        "Initial revision";
      reference
        "RFC xxxx: Adaptive Subscription to YANG Notification.";
    }

    identity adaptive-unsupported {
      base sn:establish-subscription-error;
      description
        "Adaptive-subscription is not supported for any objects
         that are selectable by the filter.";
    }

    identity xpath-evaluation-unsupported {
      base sn:establish-subscription-error;
      description
        "Unable to parse the xpath evaluation criteria defined in
         'xpath-external-eval' because a syntax error or some
         XPath 1.0 syntax not supported against the specific object.";
    }

    identity multi-xpath-criteria-conflict {
      base sn:establish-subscription-error;
      base sn:subscription-terminated-reason;
      description
        "Multiple Xpath evaluation criteria represented by
         'xpath-external-eval' is evaluated as conflict, i.e.,
         more than one condition expressions are evaluated to
         'true'.";
```

```
    }

    grouping adaptive-subscription-modifiable {
      description
        "This grouping describes the datastore-specific adaptive subscription
         conditions that can be changed during the lifetime of the
         subscription.";
      container adaptive-subscriptions {
        list adaptive-period {
          key "name";
          description
            "Defines necessary conditions to switch update interval for
             sending an event record to the subscriber. The event record output
             generation will not be influenced these conditions.";
          leaf name {
            type string {
              length "1..64";
            }
            description
              "The name of the condition to be matched.  A device MAY further
               restrict the length of this name; space and special
               characters are not allowed.";
          }
          leaf xpath-external-eval {
            type string;
            mandatory true;
            description
              "A XPath string, representing a logical expression,
               which can contain comparisons of datastore values
               and logical operations in the XPath format.";
          }
          leaf period {
            type yp:centiseconds;
            mandatory true;
            description
              "Duration of time that should occur between periodic
               push updates, in units of 0.01 seconds.";
          }
          leaf anchor-time {
            type yang:date-and-time;
            description
              "Designates a timestamp before or after which a series
               of periodic push updates are determined.  The next
               update will take place at a point in time that is a
               multiple of a period from the 'anchor-time'.
               For example, for an 'anchor-time' that is set for the
               top of a particular minute and a period interval of a
               minute, updates will be sent at the top of every
```

```
                minute that this subscription is active.";
            }
          }
          description
            "Container for adaptive subscription.";
        }
      }

      augment "/sn:subscriptions/sn:subscription/yp:update-trigger" {
        description
          "This augmentation adds additional subscription parameters
           that apply specifically to adaptive subscription.";
        case adaptive-subscriptions {
          description
            "Defines necessary conditions for sending an event record to
             the subscriber.";
          uses adaptive-subscription-modifiable;
        }
      }

      augment "/sn:establish-subscription/sn:input/yp:update-trigger" {
        description
          "This augmentation adds additional subscription parameters
             that apply specifically to datastore updates to RPC input.";
        case adaptive-subscriptions {
          description
            "Defines necessary conditions for sending an event record to
             the subscriber.";
          uses adaptive-subscription-modifiable;
        }
      }

      notification adaptive-period-update {
        sn:subscription-state-notification;
        description
          "This notification contains a push update that in turn contains
           data subscribed to via a subscription.  In the case of a
           periodic subscription, this notification is sent for periodic
           updates.  It can also be used for synchronization updates of
           an on-change subscription.  This notification shall only be
           sent to receivers of a subscription.  It does not constitute
           a general-purpose notification that would be subscribable as
           part of the NETCONF event stream by any receiver.";
        leaf id {
          type sn:subscription-id;
          description
            "This references the subscription that drove the
             notification to be sent.";
```

```
      }
      leaf period {
        type yp:centiseconds;
        mandatory true;
        description
          "New duration of time that should occur between periodic
           push updates, in units of 0.01 seconds.";
      }
      leaf anchor-time {
        type yang:date-and-time;
        description
          "Designates a timestamp before or after which a series
           of periodic push updates are determined.  The next
           update will take place at a point in time that is a
           multiple of a period from the 'anchor-time'.
           For example, for an 'anchor-time' that is set for the
           top of a particular minute and a period interval of a
           minute, updates will be sent at the top of every
           minute that this subscription is active.";
      }
      uses yp:datastore-criteria {
        refine "selection-filter/within-subscription" {
          description
            "Specifies the selection filter and where it originated
             from.  If the 'selection-filter-ref' is populated, the
             filter in the subscription came from the 'filters'
             container.  Otherwise, it is populated in-line as part
             of the subscription itself.";
        }
      }
    }
  }
  <CODE ENDS>
```

5. IANA Considerations

5.1. Updates to the IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested to be made:

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

5.2. Updates to the YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC7950]. Following the format in [RFC6020], the following registration is requested to be made:

```
---------------------------------------------------------------------
   Name:         ietf-adaptive-subscription
   Namespace:    urn:ietf:params:xml:ns:yang:ietf-adaptive-subscription
   Prefix:       as
   Reference:    RFC xxxx
---------------------------------------------------------------------
```

6. Security Considerations

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

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

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

* /sn:subscriptions/sn:subscription/yp:update-trigger/as:adaptive-subscriptions/as:adaptive-period/as:period

* /sn:subscriptions/sn:subscription/yp:update-trigger/as:adaptive-subscriptions/as:adaptive-period/as:anchor-time

* /sn:establish-subscription/sn:input/yp:update-trigger/as:adaptive-subscriptions/as:adaptive-period/as:period

* /sn:establish-subscription/sn:input/yp:update-trigger/as:adaptive-subscriptions/as:adaptive-period/as:anchor-time

7. Contributors

Thanks Michael Wang, Liang Geng for their major contributions to the initial modeling and use cases.

Michael Wang
Email: wangzitao@huawei.com

Liang Geng
China Mobile
32 Xuanwumen West St, Xicheng District
Beijing 10053

Email: gengliang@chinamobile.com

8. Acknowledges

We would like to thank Rob Wilton, Thomas Graf, Andy Bierman, Michael Richardson, Henk Birkholz for valuable review on this document, special thanks to Thomas and Michael for organizing the discussion on several relevant drafts and reach the common understanding on the concept and ideas. Thanks Michael for providing CHIP/Matter WIFI statistics reference.

9. References

9.1. Normative References

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

[RFC3198] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, DOI 10.17487/RFC3198, November 2001, <https://www.rfc-editor.org/info/rfc3198>.

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

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

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

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

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

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

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

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

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

[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>.

[RFC9196] Lengyel, B., Clemm, A., and B. Claise, "YANG Modules Describing Capabilities for Systems and Datastore Update Notifications", RFC 9196, DOI 10.17487/RFC9196, February 2022, <https://www.rfc-editor.org/info/rfc9196>.

9.2. Informative References

[CHIP] CSA, "Connected Home over IP Specification", April 2021.

[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>.

[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>.

[XPATH1.0] W3C, "https://www.w3.org/TR/1999/REC-xpath-19991116/", 11 November 1999.

Appendix A. Example YANG Module

The example YANG module used in this document represents a Wi-Fi Network Diagnostics data specified in [CHIP] which can be used by a Node to assist a user or Administrative Node in diagnosing potential problems.

YANG tree diagram for the "example-wifi-network-diagnostic" module:

```
module: example-wifi-network-diagnostic
  +--rw server
  |  +--rw bssid?                        yang:mac-address
  |  +--rw security-type?                enumeration
  |  +--rw wifi-version?                 enumeration
  |  +--rw channel-num?                  int8
  |  +--rw rssi?                         int8
  |  +--rw beacon-lost-count?            int8
  |  +--rw beacon-rx-count?              int8
  |  +--rw packet-multicast-rx-count?    int8
  |  +--rw packet-multicast-tx-count?    int8
  |  +--rw packet-unicast-rx-count?      int8
  |  +--rw packet-unicast-tx-count?      int8
  |  +--rw current-max-rate?             int8
  |  +--rw overrun-count?                int8
  +--rw events
     +--rw event* [name]
        +--rw name                    string
        +--rw disconnection?          enumeration
        +--rw association-failure?    enumeration
        +--rw connection-status?      enumeration
```

A.1. "example-wifi-mac" YANG Module

```
   module example-wifi-network-diagnostic {
     yang-version 1;
     namespace "http://example.com/yang/wifi-network-diagnostic";
     prefix wnd;

     import ietf-yang-types {
       prefix yang;
     }

     container server {
       description
         "Configuration of the WiFi Server logical entity.";
       leaf bssid {
         type yang:mac-address;
         description
           "The MAC address of a wireless access point.";
       }
       leaf security-type {
         type enumeration {
           enum unspecified {
             value 0;
           }
           enum none {
             value 1;
           }
           enum wep {
             value 2;
           }
           enum wpa {
             value 3;
           }
           enum wpa2 {
             value 4;
           }
           enum wpa3 {
             value 5;
           }
         }
         description
           "The type of Wi-Fi security used. A value of 0
            indicate that the interface is not currently
            configured or operational.";
       }
       leaf wifi-version {
         type enumeration {
           enum 80211a {
```

```
            value 0;
          }
          enum 80211b {
            value 1;
          }
          enum 80211g {
            value 2;
          }
          enum 80211n {
            value 3;
          }
          enum 80211ac {
            value 4;
          }
          enum 80211ax {
            value 5;
          }
        }
        description
          "The highest 802.11 standard version usable
           by the Node.";
      }
      leaf channel-num {
        type int8;
        description
          "The channel that Wi-Fi communication is currently
           operating on. A value of 0indicates that the interface
           is not currently configured or operational.";
      }
      leaf rssi {
        type int8;
        description
          "The RSSI of the Node's Wi-Fi radio in dBm.";
      }
      leaf beacon-lost-count {
        type int8;
        description
          "The count of the number of missed beacons the
           Node has detected.";
      }
      leaf beacon-rx-count {
        type int8;
        description
          "The count of the number of received beacons. The
           total number of expected beacons that could have been
           received during the interval since association SHOULD
           match the sum of BeaconRxCount and BeaconLostCount. ";
      }
```

```
      leaf packet-multicast-rx-count {
        type int8;
        description
          "The number of multicast packets received by
           the Node.";
      }
      leaf packet-multicast-tx-count {
        type int8;
        description
          "The number of multicast packets transmitted by
           the Node.";
      }
      leaf packet-unicast-rx-count {
        type int8;
        description
          "The number of multicast packets received by
           the Node.";
      }
      leaf packet-unicast-tx-count {
        type int8;
        description
          "The number of multicast packets transmitted by
           the Node.";
      }
      leaf current-max-rate {
        type int8;
        description
          "The current maximum PHY rate of transfer of
           data in bytes-per-second.";
      }
      leaf overrun-count {
        type int8;
        description
          "The number of packets dropped either at ingress or
           egress, due to lack of buffer memory to retain all
           packets on the ethernet network interface. The
           OverrunCount attribute SHALL be reset to 0 upon a
           reboot of the Node..";
      }
    }
    container events {
      description
        "Configuration of WIFI Network Diagnostic events.";
      list event {
        key "name";
        description
          "The list of event sources configured on the
           server.";
```

```
        leaf name {
          type string;
          description
            "The unique name of an event source.";
        }
        leaf disconnection {
          type enumeration {
            enum de-authenticated {
              value 1;
            }
            enum dis-association {
              value 2;
            }
          }
          description
            "A Node's Wi-Fi connection has been disconnected as a
             result of de-authenticated or dis-association and
             indicates the reason.";
        }
        leaf association-failure {
          type enumeration {
            enum unknown {
              value 0;
            }
            enum association-failed {
              value 1;
            }
            enum authentication-failed {
              value 2;
            }
            enum ssid-not-found {
              value 3;
            }
          }
          description
            "A Node has attempted to connect, or reconnect, to
             a Wi-Fi access point, but is unable to successfully
             associate or authenticate, after exhausting all
             internal retries of its supplicant.";
        }
        leaf connection-status {
          type enumeration {
            enum connected {
              value 1;
            }
            enum notconnected {
              value 2;
            }
```

```
            }
            description
              "A Node's connection status to a Wi-Fi network has
               changed. Connected, in this context, SHALL mean that
               a Node acting as a Wi-Fi station is successfully
               associated to a Wi-Fi Access Point.";
          }
        }
      }
    }
```

Appendix B. Adaptive Subscription and Notification Example

The examples within this document use the normative YANG module "ietf-adaptive-subscription" defined in Section 4 and the non-normative example YANG module "example-wifi-network-diagnostic" defined in Appendix A.1.

This section shows some typical adaptive subscription and notification message exchanges.

B.1. "edit-config" Example

The client configures adaptive subscription policy parameters on the server. The adaptive subscription configuration parameters require the server to support two update intervals (i.e., 5 seconds, 60 seconds) and report updates every 60 seconds if the rssi value is greater than or equal to -65dB; If the rssi value is less than -65dB, switch to 5 seconds period value to report updates.

```
<rpc message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
  <target>
   <running/>
  </target>
  <config
   xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
   <top
    xmlns="http://example.com/schema/1.2/config"
    xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
     xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
            ds:running
    </yp:datastore>
    <yp:datastore-xpath-filter
     xmlns:wnd="https://example.com/sample-data/1.0">
           /wnd:example-wifi-network-diagnostic
    </yp:datastore-xpath-filter>
    <as:adaptive-subscriptions
     xmlns:as="urn:ietf:params:xml:ns:yang:ietf-adaptive-subscription">
     <as:adaptive-period>
      <as:xpath-external-eval>
       /wnd:server/wnd:rssi&lt;-65
      </as:xpath-external-eval>
      <as:period>5</as:period>
     </as:adaptive-period>
     <as:adaptive-period>
      <as:xpath-external-eval>
       /wnd:server/wnd:rssi&gt;=-65
      </as:xpath-external-eval>
      <as:period>60</as:period>
    </as:adaptive-period>
    </as:adaptive-subscriptions>
   </top>
  </config>
 </edit-config>
</rpc>
```

B.2. Create Adaptive Subscription Example

The subscriber sends an "establish-subscription" RPC with the parameters listed in to request the creation of an adaptive subscription. The adaptive subscription configuration parameters require the server to report updates every 5 seconds if the rssi value is less than -65dB; If the rssi value is greater than or equal to -65dB, switch to 60 seconds period value. (Section 2)

```
<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:running
  </yp:datastore>
  <yp:datastore-xpath-filter
   xmlns:wnd="https://example.com/sample-data/1.0">
       /wnd:example-wifi-network-diagnostic
  </yp:datastore-xpath-filter>
  <as:adaptive-subscriptions
   xmlns:as="urn:ietf:params:xml:ns:yang:ietf-adaptive-subscription">
   <as:adaptive-period>
    <as:xpath-external-eval>
     wnd:server/wnd:rssi&lt;-65
    </as:xpath-external-eval>
    <as:period>5</as:period>
   </as:adaptive-period>
   <as:adaptive-period>
    <as:xpath-external-eval>
     wnd:server/wnd:rssi&gt;=-65
    </as:xpath-external-eval>
    <as:period>60</as:period>
   </as:adaptive-period>
  </as:adaptive-subscriptions>
 </establish-subscription>
</netconf:rpc>
```

B.3. "xpath-evaluation-unsupported" error response example

If the subscriber has authorization to establish the subscription with a server, but the server had not been able to fully satisfy the request from the subscriber, the server should send an RPC error response.

For instance, if the XPATH 1.0 syntax against the targeted data object defined in "xpath-external-eval" is not supported by the server' s implementation, the server returns a reply indicating a failure. The following <rpc-reply> illustrates an example:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>
      ietf-adaptive-subscription:xpath-evaluation-unsupported
    </error-app-tag>
    <error-path xmlns:wnd="https://example.com/sample-data/1.0">
      /wnd:server/wnd:rssi
    </error-path>
  </rpc-error>
</rpc-reply>
```

Since adaptive subscription allows a server to be configured with multiple different period intervals and corresponding XPath evaluation criteria to trigger update interval switch in the server, it may be possible for the server to return multiple <rpc-error> elements with "xpath-evaluation-unsupported" failure specified by different error paths. The subscriber can use this information in future attempts to establish a subscription.

B.4. "adaptive-period-update" notification example

Upon the server switches from the update interval 5 seconds to the new update interval 60 seconds, before sending event records to receivers, the "adaptive-period-update" notification should be generated and sent to the receivers to inform the receivers that the update interval value is switched to the new value.

```
<notification
 xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0"
 xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
 <eventTime>2016-11-21T13:51:00Z</eventTime>
 <adaptive-period-update
  xmlns="http://example.com/ietf-adaptive-subscription">
  <id>0</id>
  <period>60</period>
  <yp:datastore
   xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
       ds:running
  </yp:datastore>
  <yp:datastore-xpath-filter
   xmlns:ex="https://example.com/sample-data/1.0">
       /ex:example-wifi-network-diagnostic
  </yp:datastore-xpath-filter>
 </adaptive-period-update>
</notification>
```

B.5. Changes between Revisions

v00 -v01

* Clarify what if multiple Xpath condition expressions conflict with each other during the lifecycle of an adaptive subscription

* Clarify that existing RPC errrors defined in RFC 8639 and 8641 are still supported by this document

* Refine the YANG module: add contact information, fix IETF Trust Copyright statement, fix yanglint validation error

v09 -v10

* Change the draft intended status to "experimental"

* Problem statement refinement

v08 -v09

* Define two new RPC errors to report when adaptive subscription unsupported or multiple XPath criteria conflict.

* Remove the "watermark" parameter.

* Add clarification about how to evaluate the XPath expression defined in "xpath-external-eval".

* Add clarification about how to compare a targeted data object in a specific list entry.

v07 -v08

* Define a new RPC error to report when an XPath syntax defined in "xpath-external-eval" is unsupported by a server.

* Add a new example showing how the RPC error being returned by a publisher.

* The usage examples fixed in the Appendix.

* Grammatical errors correction(missing articles, plurality mismatches, etc).

v06 -v07

* The usage examples typo fixed in the Appendix.

* Add reference to RFC7950 XPATH Evaluation section and XPATH 1.0

* Clarify the definitions of 'xpath-external-eval' and 'selection-filter' by reusing XPATH Evaluation rules in RFC7950.

* Add a new terminology "adaptive subscription".

* Add one section to discuss Arbitrary XPath Complexity.

v05 -v06

* Replace example-wifi-mac module with example-wifi-network-diagnostic using WIFI statistics specified in CHIP specification.

* Update adaptive subscription Example to align with WIFI example module change.

* Add one more reference to CHIP Specification.

v04 -v05

* Remove "modify-subscption" RPC usage.

* Module update to fix the nits.

* Update adaptive subscription Example.

* Other Editorial changes.

v03 - v04

* Add missing subtrees and data nodes in the security section;

* Change "adaptive-update" notification into "adaptive-period-update" notification;

* Other Editorial changes.

v02 - v03

* Clarify the difference between low priority telemetry data dropping and collection rate switching in the introduction section;

* Update the abstract and introduction section to focus on collection rate switching in the server without interaction with the remote client;

* Format usage example and change ssid into rssi in the appendix;

* Use boilerplate and reuse the terms in the terminology section.

Authors' Addresses

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

Wei Song
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: songwei80@huawei.com

Peng Liu
China Mobile
32 Xuanwumen West St, Xicheng District
Beijing
Email: liupengyjy@chinamobile.com

Qiufang Ma
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: maqiufang1@huawei.com

Wei Wang
China Telecom
32 Xuanwumen West St, Xicheng District
Beijing
Email: wangw36@chinatelecom.cn

Zhixiong Niu
Microsoft
Email: Zhixiong.Niu@microsoft.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.

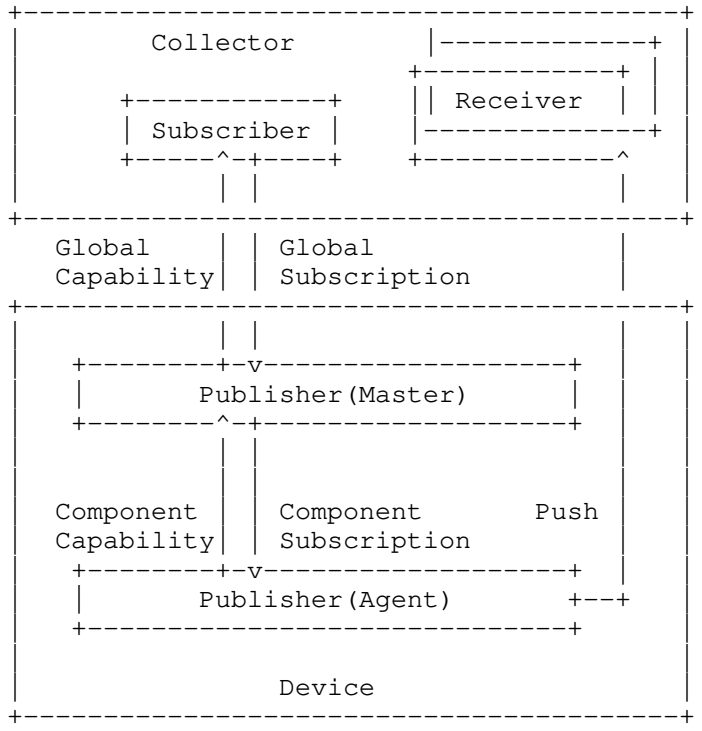

Figure 1: Fig. 2 The Distributed Data Export Framework

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.

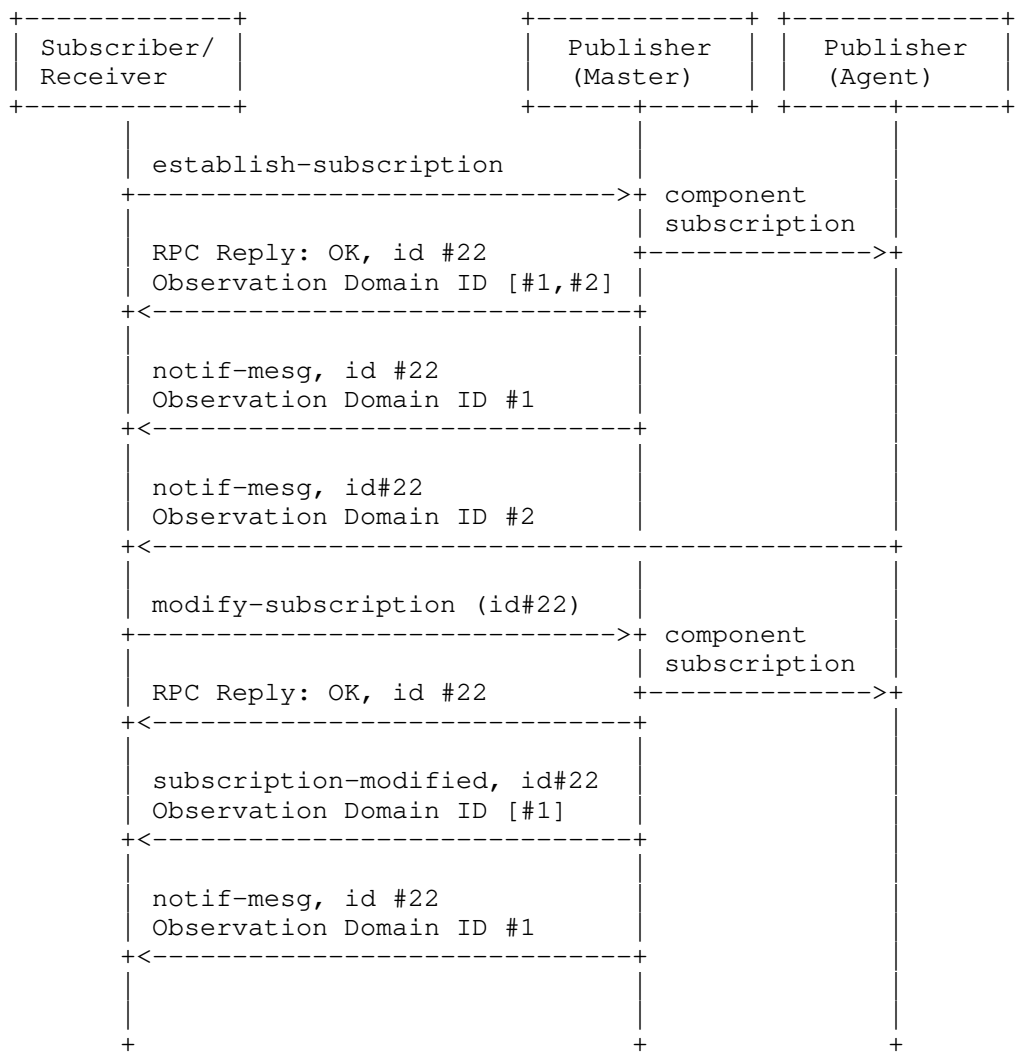

Figure 2: Fig. 3 Call Flow for Dynamic Subscription

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.

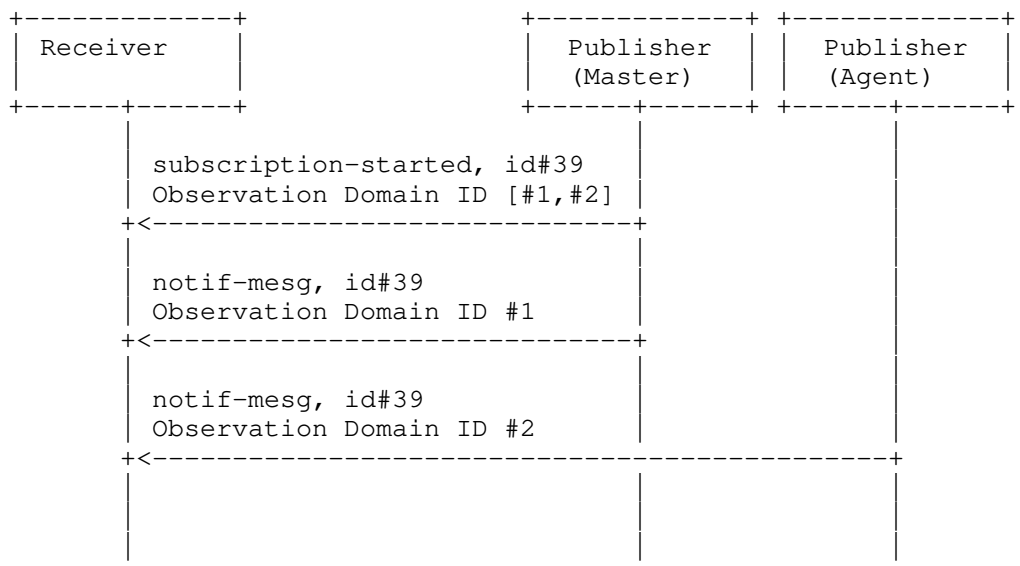

Figure 7: Fig. 8 Call Flow for Configured Subscription

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

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

S. Turner
sn3rd
R. Housley
Vigil Security

24 October 2022

NETCONF over TLS 1.3
draft-ietf-netconf-over-tls13-01

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.

About This Document

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

The latest revision of this draft can be found at https://netconf-wg.github.io/netconf-over-tls13/draft-ietf-netconf-over-tls13.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-netconf-over-tls13/.

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

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

Status of This Memo

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. Note that TLS 1.3 can be used without early data as per Appendix F.5 of [I-D.ietf-tls-rfc8446bis]. In fact, early data is permitted by TLS 1.3 only when the client and server share a Pre-Shared Key (PSK), either obtained externally or via a previous handshake. The client uses the PSK to authenticate the server and to encrypt the early data.

As noted in Section 2.3 of [I-D.ietf-tls-rfc8446bis], the security properties for early data are weaker than those for subsequent TLS-protected data. In particular, early data is not forward secret, and there is no protection against the replay of early data between connections. Appendix E.5 of [I-D.ietf-tls-rfc8446bis] requires applications not use early data without a profile that defines its use. This document specifies that NETCONF implementations that support TLS 1.3 MUST NOT use early data.

4. Cipher Suites

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

Implementations that support TLS 1.3 MAY implement additional TLS cipher suites that provide mutual authentication and confidentiality, which are required for NETCONF [RFC6241].

NETCONF 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 incorrectly identified peer; see Section 7 of [RFC7589]. If deployments make use of a trusted 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].

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-11, 16 August 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-uta-rfc7525bis-11>.

[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>.

[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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2023

JG. Cumming
Nokia
R. Wills
Cisco Systems

21 October 2022

NETCONF Private Candidates
draft-jgc-netconf-privcand-00

Abstract

This document provides a mechanism to extend the Network Configuration Protocol (NETCONF) to support multiple clients making configuration changes simultaneously and ensuring that they commit only those changes that they defined.

This document addresses two specific aspects: The interaction with a private candidate over the NETCONF protocol and the methods to identify and resolve conflicts between clients.

Table of Contents

1. Introduction

NETCONF [RFC6241] provides a mechanism for one or more clients to make configuration changes to a device running as a NETCONF server. Each NETCONF client has the ability to make one or more configuration change to the servers shared candidate configuration.

As the name shared candidate suggests, all clients have access to the same candidate configuration. This means that multiple clients may make changes to the shared candidate prior to the configuration being committed. This behaviour may be undesirable as one client may unwittingly commit the configuration changes made by another client.

NETCONF provides a way to mitigate this behaviour by allowing clients to place a lock on the shared candidate. The placing of this lock means that no other client may make any changes until that lock is released. This behaviour is, in many situations, also undesirable.

Many network devices already support private candidates configurations, where a user (machine or otherwise) is able to edit a personal copy of a devices configuration without blocking other users from doing so.

This document details the extensions to the NETCONF protocol in order to support the use of private candidates.

1.1. Requirements Language

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

2. Definitions and terminology

2.1. Session specific datastore

A session specific datastore is a configuration datastore that, unlike the candidate and running configuration datastores which have only one per system, is bound to the specific NETCONF session.

2.2. Shared candidate configuration

The candidate configuration datastore defined in [RFC6241] is referenced as the shared candidate configuration in this document.

2.3. Private candidate configuration

A private candidate configuration is a session specific candidate configuration datastore.

The specific NETCONF session (and user) that created the private candidate configuration is the only session (user) that has access to it over NETCONF. Devices may expose this to other users through other interfaces but this is out of scope for this document.

The private candidate configuration contains a copy of the running configuration when it is created (in the same way as a branch does in a source control management system). Any changes made to it, for example, through the use of the <edit-config> operation, are made in this private candidate configuration. Obtaining this private candidate over NETCONF will display the entire configuration, including all changes made to it. Performing a <commit> operation will merge the changes from the private candidate into the running configuration (the same as a merge in source code management systems). The issue of <discard-changes> operation will revert the private candidate to the branch's initial state.

All changes made to this private candidate configuration are held separately from any other candidate configuration changes, whether made by other users to the shared candidate or any other private candidate, and are not visible to or accessible by anyone else.

3. Limitations using the shared candidate configuration for multiple clients

The following sections describe some limitations and mitigation factors in more detail for the use of the shared candidate configuration during multi-client configuration over NETCONF.

3.1. Issues

3.1.1. Unintended deployment of alternate users configuration changes

Consider the following scenario:

1. Client 1 modifies item A in the shared candidate configuration

2. Client 2 then modifies item B in the shared candidate configuration

3. Client 2 then issues a <commit> RPC

In this situation, both client 1 and client 2 configurations will be committed by client 2. In a machine-to-machine environment client 2 may not have been aware of the change to item A and, if they had been aware, may have decided not to proceed.

3.2. Current mitigation strategies

3.2.1. Locking the shared candidate configuration datastore

In order to resolve unintended deployment of alternate users configuration changes as described above NETCONF provides the ability to lock a datastore in order to restrict other users from editing and committed changes.

This does resolve the specific issue above, however, it introduces another issue. Whilst one of the clients holds a lock, no other client may edit the configuration. This will result in the client failing and having to retry. Whilst this may be a desirable consequence when two clients are editing the same section of the configuration, where they are editing different sections this behaviour may hold up valid operational activity.

Additionally, a lock placed on the shared candidate configuration must also lock the running configuration, otherwise changes committed directly into the running datastore may conflict.

3.2.2. Always use the running configuration datastore

The use of the running configuration datastore as the target for all configuration changes does not resolve any issues regarding blocking of system access in the case a lock is taken, nor does it provide a solution for multiple NETCONF clients as each configuration change is applied immediately and the client has no knowledge of the current configuration at the point in time that they commenced the editing activity nor at the point they commit the activity.

3.2.3. Fine-grained locking

[RFC5717] describes a partial lock mechanism that can be used on specific portions of the shared candidate datastore.

Partial locking does not solve the issues of staging a set of configuration changes such that only those changes get committed in a commit operation, nor does it solve the issue of multiple clients editing the same parts of the configuration at the same time.

Partial locking additionally requires that the client is aware of any interdependencies within the servers YANG models in order to lock all parts of the tree.

4. Key choices influencing the solution

This section captures the key aspects considered when defining the private candidate solution.

4.1. When is a private candidate created

A private candidate datastore is created when the first RPC that requires access to it is sent to the server. This could be, for example, an <edit-config>.

When the private candidate is created is copy of the running configuration is made and stored in it. This can be considered the same as creating a branch in a source code repository.

4.2. Interaction between running and private-candidate

Multiple NETCONF operations may be performed on the private candidate in order to stage changes ready for a commit.

A key consideration is how and when the private candidate is updated by changes made to the running configuration whilst the private candidate (a separate branch) exists.

The following options have been considered. It is worth noting that both approaches may be supported, however, the server will need to advertise which approach is being used in a capability.

4.2.1. Independent private candidate branch (Static branch mode)

The private candidate is treated as a separate branch and changes made to the running configuration are not placed into the private candidate datastore except in one of the following situations:

* The client requests that the private candidate be refreshed using a new <update> operation

* <commit> is issued

* <discard-changes> operation is sent (TBD).

This approach is similar to the standard approach for source code management systems.

In this model of operation it is possible for the private candidate configuration to become significantly out of sync with the running configuration should the private candidate be open for a long time without an operation being sent that causes a resync (rebase in source code control terminology).

A <compare> operation may be performed against the initial starting point (head) of the private candidates branch or against the running configuration.

Conflict detection and resolution is discussed later in this document.

4.2.2. Continually updating private candidate (Continuous rebase mode)

The private candidate is treated as a separate branch, however, changes made to the running configuration and reflected in the private candidate configuration as they occur.

This is equivalent to the private candidate branch being routinely rebased onto the running configuration every time a change is made in the running configuration.

In this model of operation the following should be considered:

* Because the private candidate is automatically re-synchronized (rebased) with the running configuration each time a change is made in the running configuration, the NETCONF session is unaware that their private candidate configuration has changed unless they perform one of the get operations on the private candidate and analyse it for changes.

* A <compare> operation may be performed against the initial starting point (head) of the private candidates branch or against the running configuration but these will both report the same results as the starting point is continually reset.

* The output of the <compare> operation may not match the set of changes made to the session's private candidate but may include different output due to the changes in the running configuration made by other sessions.

* A conflict may occur in the automatic update process pushing changes from the running configuration into the private candidate.

Conflict detection and resolution is discussed later in this document.

4.3. Defining and detecting conflicts

The most challenging aspect of private candidates is when two clients are modifying the same part of the configuration tree.

A conflict occurs when a private candidate configuration is committed to the running configuration datastore and the specific nodes in the tree to be modified have been changed in the running configuration after the private candidate was created.

If using the continual rebase mode, a conflict may also occur if a specific node (or set of nodes) in the modified private candidate configuration are updated by another client (or user) in the running configuration.

Conflicts occur when the intent of the NETCONF client may have been different had it had a different starting point. When a conflict occurs it is useful that the client be given the opportunity to re-evaluate its intent. Examples of conflicts include:

* An interface has been deleted in the running configuration that existed when the private candidate was created. A change to a child node of this specific interface is made in the private candidate using the default merge operation would, instead of changing the child node, both recreate the interface and then set the child node.

* A leaf has been modified in the running configuration from the value that it had when the private candidate was created. The private candidate configuration changes that leaf to another value.

4.4. Reporting unresolved conflicts to the user

When a conflict is detected the <commit> MUST fail with a specific error message and the client SHOULD be informed which conflicts caused the failure.

There are two ways conflicts could be reported:

* Using an attribute on the data node(s) that have conflicts.

* As a list of flat paths (similar to how errors from a commit operation are reported).

4.5. Resolving conflicts

There are different options for resolving conflicts:

* The user could be required to explicitly resolve all conflicts by performing further operations to the private candidate.

* The private candidate could take precedence (equivalent to a <force> option).

* The running config could take precedence (for example, by cancelling changes in the private candidate if they conflict with changes already made to the running config).

5. Proposed solutions for using private candidates configurations with NETCONF

NETCONF sessions are able to utilize the concept of private candidates in order to streamline network operations, particularly for machine-to-machine communication.

Using this approach clients may improve their performance and reduce the likelihood of blocking other clients from continuing with valid operational activities.

One or more private candidates may exist at any one time, however, a private candidate MUST:

* Be accessible by one client only

* Be visible by one client only

Additionally, the choice of using a shared candidate configuration datastore or a private candidate configuration datastore SHOULD be for the entire duration of the NETCONF session

The options provided below are not intended to be mutually exclusive and multiple options may be supported by the server.

5.1. Client capability declaration

When a NETCONF client connects with a server it sends a list of client capabilities.

In order to enable private candidate mode for the duration of the NETCONF client session the NETCONF client sends the following capability:

urn:ietf:netconf:pc

The ability for the NETCONF server to support private candidates is optional and SHOULD be signalled in the NETCONF servers capabilities using the same capability string

When a server receives the client capability its mode of operation will be set to private candidates for the duration of the NETCONF session.

When a client makes a configuration change the <edit-config> RPC will target the candidate datastore as it does in shared candidate configuration mode.

All RPCs will operate in an identical manner to when operating in shared candidate configuration mode but all data sent between the client and the candidate datastore will use that sessions private candidate configuration.

Using this method, the use of private candidates can be made available to NMDA and non-NMDA capable servers.

No protocol extensions are required for the transitioning of candidates between the shared mode and the private mode and no extensions are required for the any other RPC (including <lock>)

5.2. Private candidate datastore

The private candidate configuration datastore could be exposed as its own datastore similar to other NMDA [RFC8342] capable datastores. This datastore is called private-candidate.

All NMDA operations that support NMDA datastores SHOULD support the private-candidate datastore.

Any non-NMDA aware NETCONF operations that take a source or target (destination) may be extended to accept the new datastore.

The ability for the NETCONF server to support private candidates is optional and SHOULD be signalled in NMDA supporting servers as a datastore and in all NETCONF servers capabilities using the capability string:

urn:ietf:netconf:pc

The first datastore referenced (either candidate or private-candidate) in any NETCONF operation will define which mode that NETCONF session will operate in for its duration. As an example, performing a <get-data> operation on the private-candidate datastore will switch the session into private candidate configuration mode and subsequent <edit-config> operations that reference the candidate configuration datastore will fail.

5.2.1. New and existing NETCONF operation interactions

This section mentions a small number of operations whose behaviour is needs a special mention, other operations to be updated are detailed in the appendix.

5.2.1.1. <update>

The new <update> operation is provided in order to trigger the private-candidate configuration datastore to be updated (rebased in source code management terminology) with the changes from the running configuration.

5.2.1.2. <edit-config>

The <edit-config> operation is updated to accept private-candidate as valid input to the <target> field.

The use of <edit-config> will create a private candidate configuration if one does not already exist for that NETCONF session.

Sending an <edit-config> request to private-candidate after one has been sent to the shared candidate datastore in the same session will fail (and visa-versa).

Multiple <edit-config> requests may be sent to the private-candidate datastore in a single session.

5.2.1.3. <lock> and <unlock>

Performing a <lock> on the private-candidate datastore is a valid operation and will also lock the running configuration.

Taking a lock on this datastore will stop other session from committing any configuration changes, regardless of the datastore.

Other NETCONF sessions are still able to create a new private-candidate configuration datastore.

Performing an <unlock> on the private-candidate datastore is a valid operation. This will also unlock the running configuration. Unlocking the private-candidate datastore allows other sessions to resume <commit> functions.

Changes in the private-candidate datastore are not lost when the lock is released.

Attempting to perform a <lock> or <unlock> on any other datastore while the private-candidate datastore is locked will fail. Attempting to perform a <lock> or <unlock> on any other sessions private-candidate datastore will also fail.

5.2.1.4. <compare>

Performing a <compare> [RFC9144] with the private-candidate datastore as either the <source> or <target> is a valid operation.

If <compare> is performed prior to a private candidate configuration being created, one will be created at that point.

The <compare> operation will be extended to allow the operation to reference the start of the private candidate's branch (head).

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

This document should not affect the security of the Internet.

8. References

8.1. Normative References

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

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

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

[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>.

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

[RFC5717] Lengyel, B. and M. Bjorklund, "Partial Lock Remote Procedure Call (RPC) for NETCONF", RFC 5717, DOI 10.17487/RFC5717, December 2009, <https://www.rfc-editor.org/info/rfc5717>.

8.2. Informative References

Appendix A. NETCONF operations impacted

A.1. <get>

The <get> operation does not accept a datastore value and therefore this document is not applicable to this operation. The use of the get operation will not create a private candidate configuration.

A.2. <get-config>

The <get-config> operation is updated to accept private-candidate as valid input to the <source> field.

The use of <get-config> will create a private candidate configuration if one does not already exist for that NETCONF session.

A.3. <get-data>

The <get-data> operation accepts the private-candidate as a valid datastore.

The use of <get-data> will create a private candidate configuration if one does not already exist for that NETCONF session.

A.4. <copy-config>

The <copy-config> operation is updated to accept private-candidate as a valid input to the <source> or <target> fields.

A.5. <delete-config>

The <delete-config> operation is updated to accept private-candidate as a valid input to the <target> field.

Contributors

The authors would like to thank Jan Lindblad, Jason Sterne and Rob Wilton for their contributions and reviews.

Authors' Addresses

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

Robert Wills
Cisco Systems
Email: rowills@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.

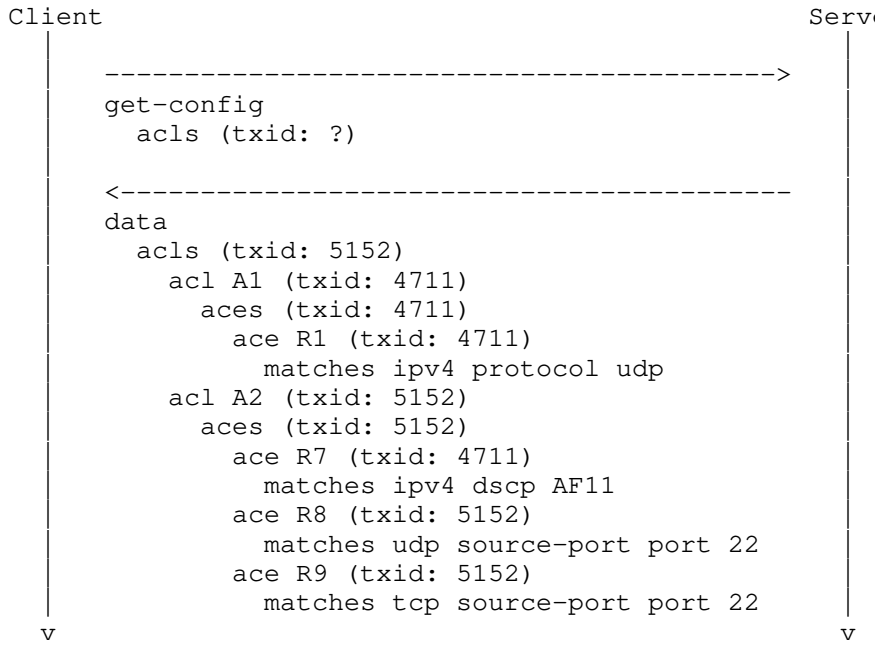

Figure 1: Initial Configuration Retrieval. The server returns the requested configuration, annotated with txid values. The most recent change seems to have been an update to the R8 and R9 source-port.

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.

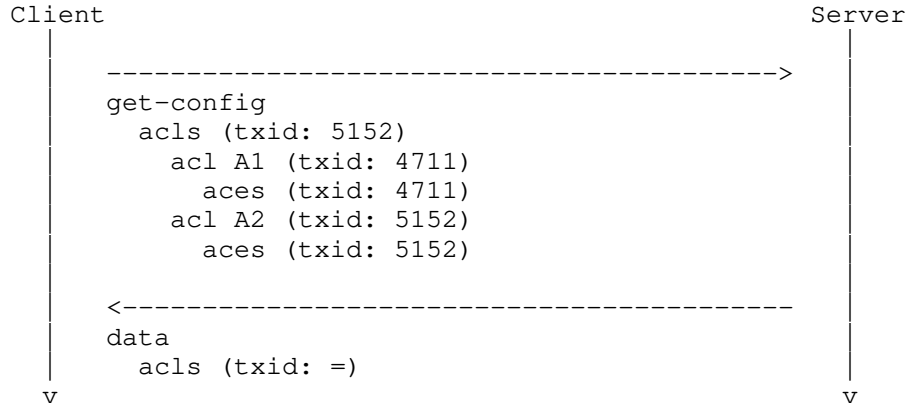

Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where txid matches expectations.

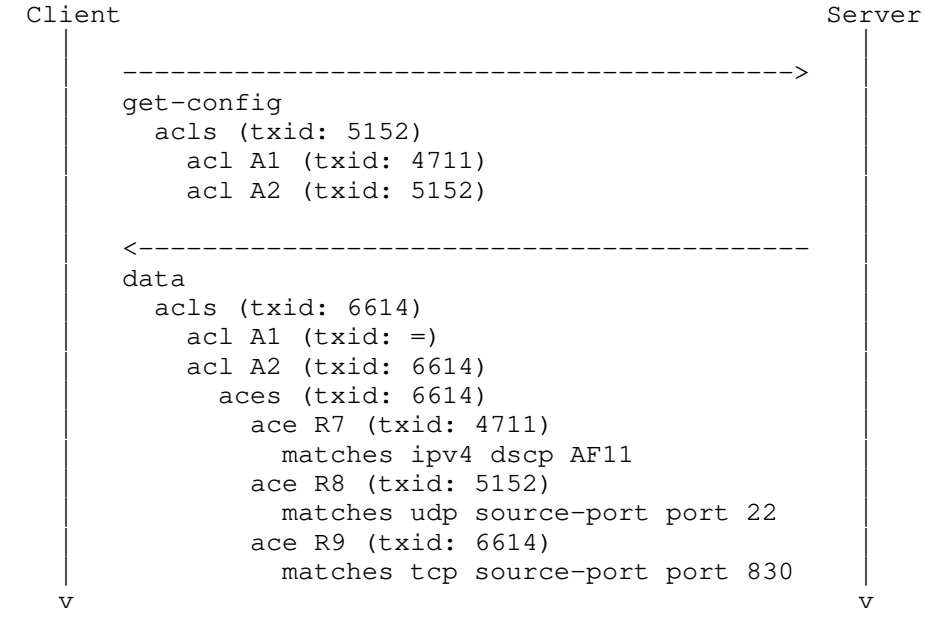

Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides update where changes have happened.

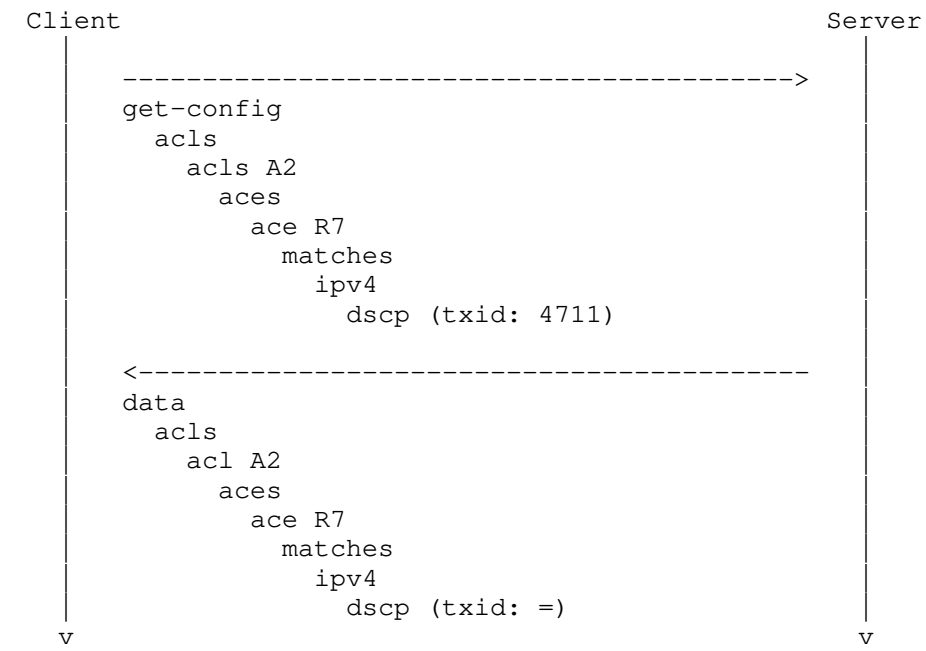

Figure 4: Versioned nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the etag value is '=', and the leaf value is pruned.

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.

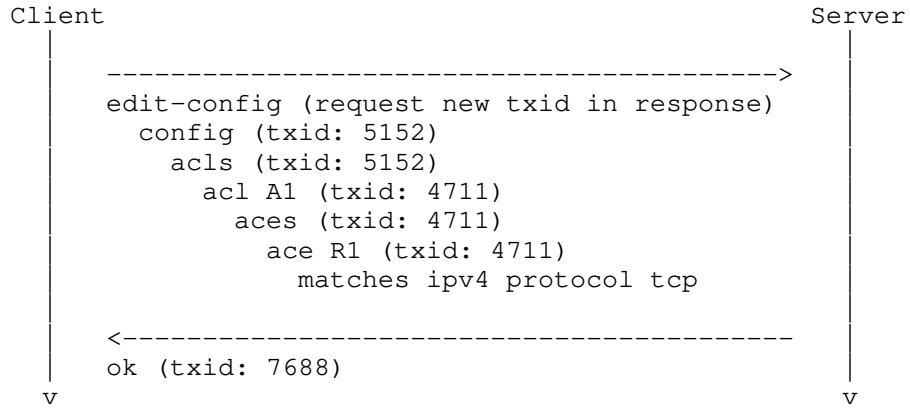

Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

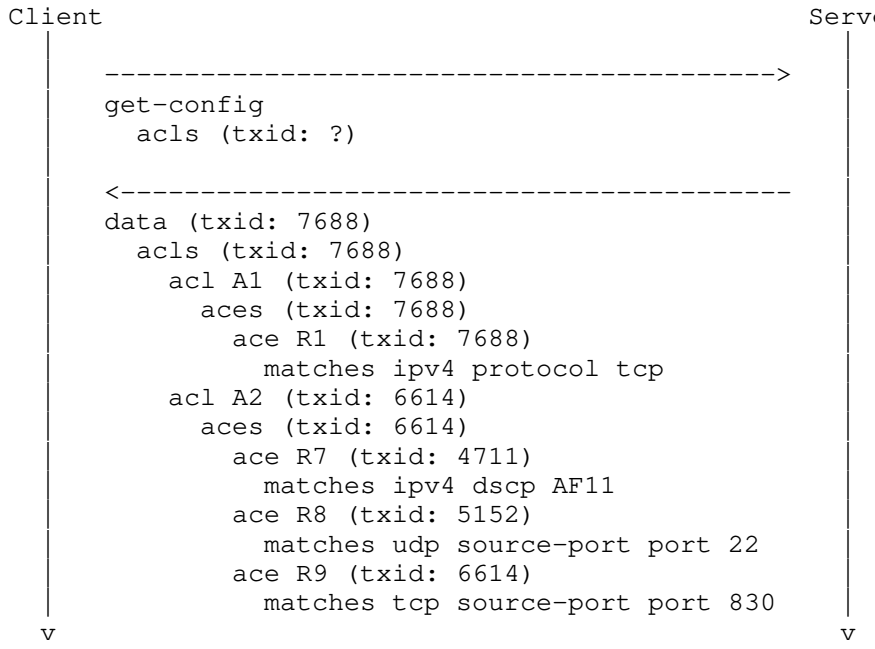

Figure 6: For all leaf objects that were changed, and all their ancestors, the txids are updated to the value returned in the ok message.

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.

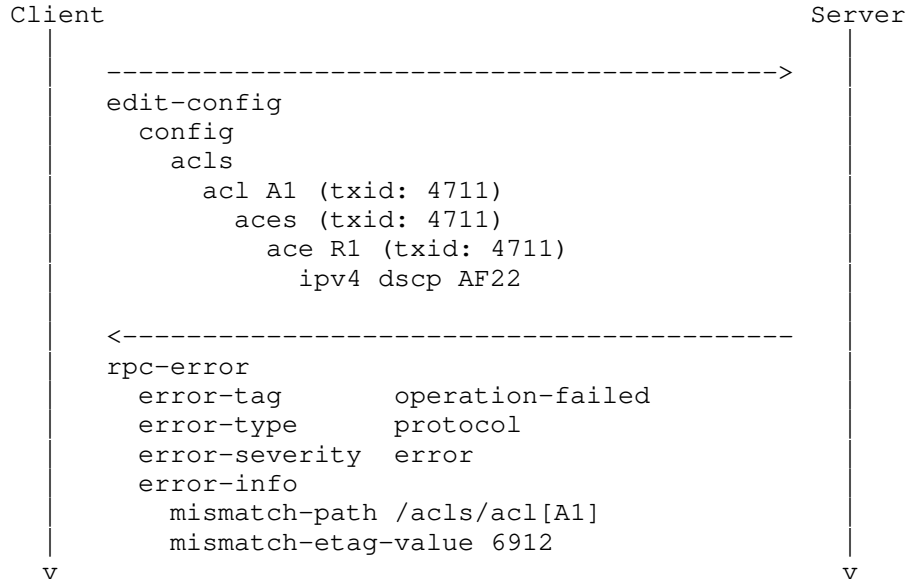

Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the txid it knows for this part of the configuration. Since the txid has changed (out of band), the server rejects the configuration 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.

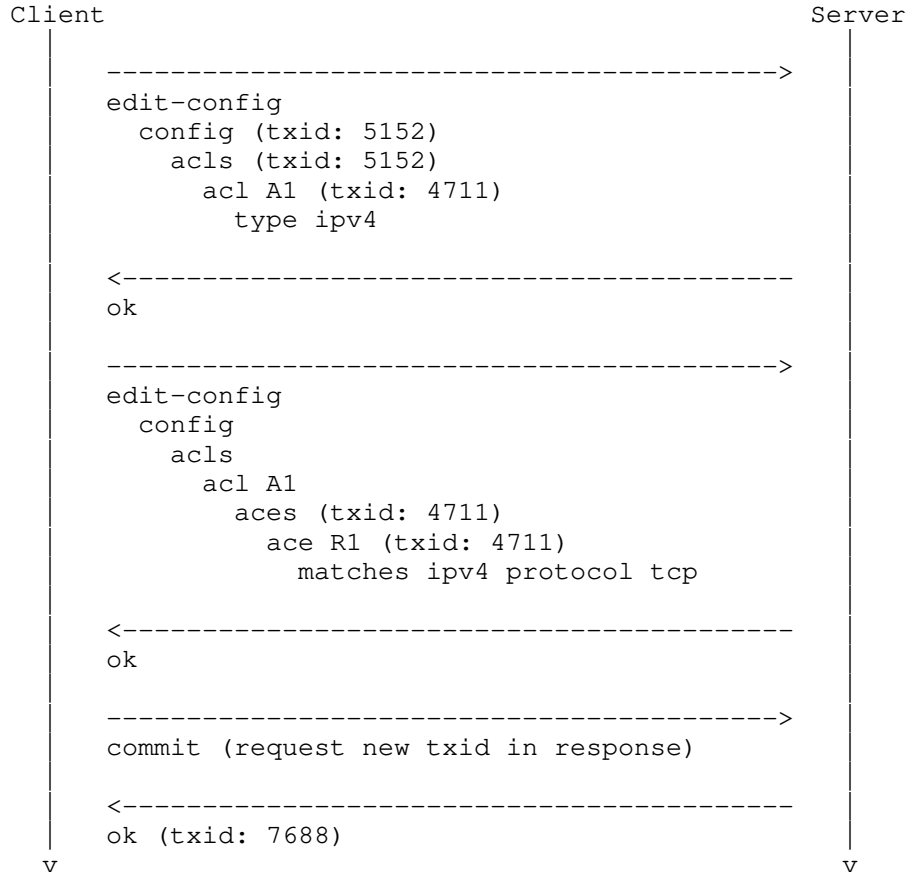

Figure 8: Conditional transaction towards the Candidate datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

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:

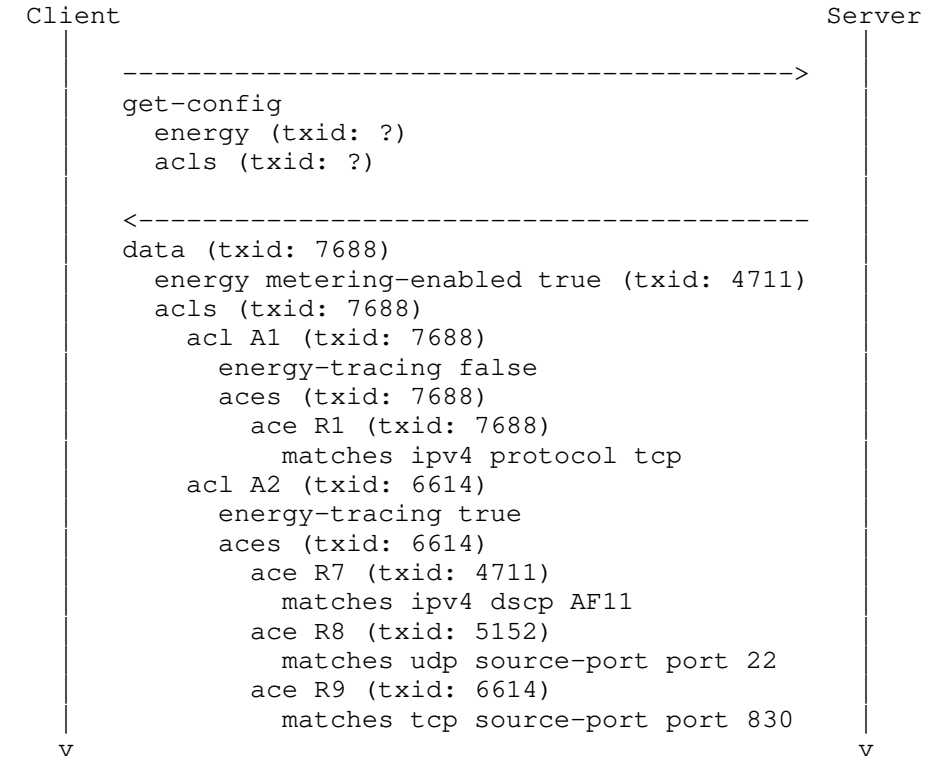

Figure 9: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

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.

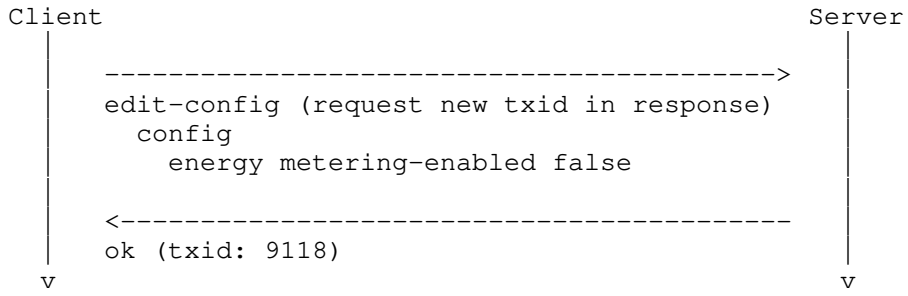

Figure 10: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in txid changes.

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

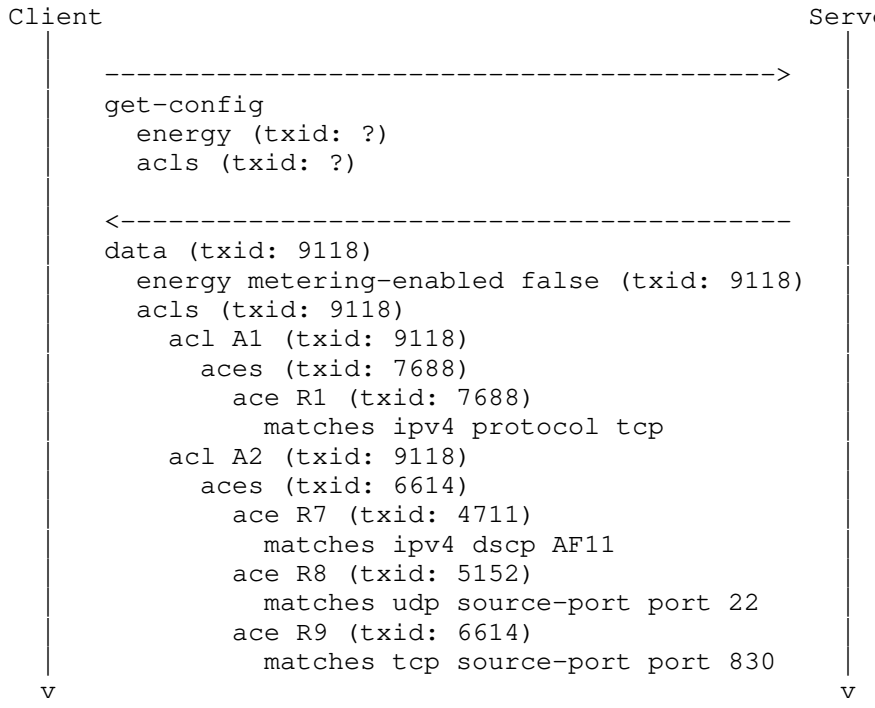

Figure 11: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when- expression.

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

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2023

J. Quilbeuf
B. Claise
Huawei
T. Graf
Swisscom
D. Lopez
Telefonica I+D
Q. Sun
China Telecom
20 October 2022

External Transaction ID for Configuration Tracing
draft-quilbeuf-opsawg-configuration-tracing-00

Abstract

Network equipments are often configured by a variety of network management systems (NMS), protocols, and people. If a network issue arises because of a wrong configuration modification, it's important to quickly identify the specific service request and obtain the reason for pushing that modification. Another potential network issue can stem from concurrent NMS's with overlapping intent, each having their own tasks to perform: in such a case, it's important to map the respective modifications to its originating NMS. This document specifies a mechanism to automatically map the configuration modifications to their source, up to a specific NMS service request, in the context of NETCONF. Such a mechanism is required for autonomous networks, to trace the reason of a particular configuration change that lead to an anomaly detection or a broken SLA. This mechanism facilitates the troubleshooting, the post mortem analysis, and in the end the closed loop automation required for self-healing networks. The specifications contain a new YANG module mapping a local configuration change to the corresponding northbound transaction, up to the controller or even the orchestrator.

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/JeanQuilbeufHuawei/draft-quilbeuf-opsawg-configuration-tracing.

Status of This Memo

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

This Internet-Draft will expire on 23 April 2023.

Table of Contents

1. Introduction

Issues arising in the network, for instance violation of some SLAs, might be due to some configuration modification. In the context of automated networks, the assurance system needs not only to identify and revert the problematic configuration modification, but also to make sure that it won't happen again and that the fix will not disrupt other services. To cover the last two points, it is imperative to understand the cause of the problematic configuration change. Indeed, the first point, making sure that the configuration modification will not be repeated, cannot be ensured if the cause for pushing the modification in the first place is not known. Ensuring the second point, not disrupting other services, requires as well knowing if the configuration modification was pushed in order to support new services. Therefore, we need to be able to trace a configuration modification on a device back to the reason that triggered that modification, for instance in a NMS, whether the controller or the orchestrator.

This specification focuses only on configuration pushed via NETCONF [RFC6241]. The rationale for this choice is that NETCONF is better suited for normalization than other protocols (SNMP, CLI). Another reason is that the notion of transaction ID, useful to track configuration modification, is already defined in [I-D.lindblad-netconf-transaction-id] and comes from RESTCONF [RFC8040].

The same network element, or NETCONF [RFC6241] server, can be configured by different NMSs or NETCONF clients. If an issue arises, one of the starting points for investigation is the configuration modification on the devices supporting the impacted service. In the best case, there is a dedicated user for each client and the timestamp of the modification allows tracing the problematic modification to its cause. In the worst case, everything is done by the same user and some more tricks must be done to trace the problematic modification to its source.

This document specifies a mechanism to automatically map the configuration modifications to their source, up to a specific NMS service request. Practically, this mechanism annotates configuration changes on the configured element with sufficient information to unambiguously identify the corresponding transaction, if any, on the

element that requested the configuration modification. It reuses the concept of a NETCONF transaction ID from [I-D.lindblad-netconf-transaction-id] and augment it with an ID for the client. The information needed to do the actual configuration tracing is stored in a new YANG module that maps a local configuration change to the corresponding northbound transaction, up to the controller or even the orchestrator. In case of a controller, the local configuration modification ID to both corresponding north- and southbound transaction ID. Additionally, for northbound transactions, we store the ID of the client.

2. Terminology

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

This document uses the terms client and server from [RFC6241].

This document uses the terms transaction and transaction id from [I-D.lindblad-netconf-transaction-id].

3. Use cases

This document was written with autonomous networks in mind. We assume that an existing monitoring or assurance system, such as described in [I-D.ietf-opsawg-service-assurance-architecture], is able to detect and report network anomalies , e.g. SLA violations, intent violations, network failure, or simply a customer issue. Here are the use cases for the proposed YANG module.

3.1. Configuration Mistakes

Taking into account that many network anomalies are due to configuration mistakes, this mechanism allows to find out whether the offending configuration modification was triggered by a tracing-enabled client/NMS. In such as case, we can map the offending configuration modification id on a server/NE to a local configuration modification id on the client/NMS. Assuming that this mechanism (the YANG module) is implemented on the controller, we can recursively find, in the orchestrator, the latest (set of of) service request(s) that triggered the configuration modification. Whether this/those service request(s) are actually the root cause needs to be investigated. However, they are a good starting point for troubleshooting, post mortem analysis, and in the end the closed loop automation, which is absolutely required for for self-healing

networks.

3.2. Concurrent NMS Configuration

Building on the previous use case is the situation where two NMS', unaware of the each other, configuring routers, each believing that they are the only NMS for specific device. So one configuration executed by the NMS1 is overwritten by the NMS2, which in turn is overwritten by NMS1, etc.

3.3. Conflicting Intents

Autonomous networks will be solved first by assuring intent per specific domain; for example data center, core, cloud, etc. This last use case is a more specific "Concurrent NMS configuration" use case where assuring domain intent breaks the entire end to end service, even if the domain-specific controllers are aware of each other.

4. Relying on Transaction-id to Trace Configuration Modifications

4.1. Instantiating the YANG module

In [I-D.lindblad-netconf-transaction-id], the concept of a NETCONF transaction ID is proposed, to match the same mechanism from RESTCONF [RFC8040]. The goal of this document is to speed up the re-synchronization process between a client and a server, by using a common transaction ID. If the current transaction ID on the server is the same as the transaction ID known by the client, then both are synchronized. Otherwise, the client has to fetch again the configuration. The transaction ID can be applied to the whole configuration or to so-called versioned nodes. In the latter case, only versioned nodes for which the transaction ID differs need to be updated.

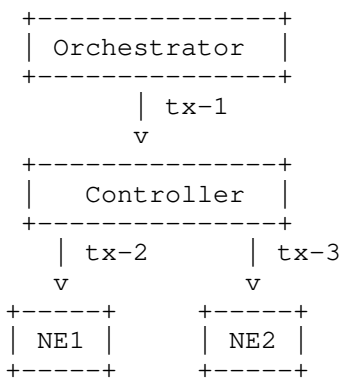

Figure 1: Example of Hierarchical Configuration. tx: transaction

A server considers as a northbound transaction a transaction that modifies its configuration. On Figure 1, tx-1 is a northbound transaction for the Controller.

A client considers as a southbound transaction the modification of a server configuration. On Figure 1, tx-2 and tx-3 are southbound transactions for the Controller.

If the set-tx-id feature is enabled (see open issue in Section 9.1), the client can specify its own transaction ID when sending the configuration ID for the server. In that case, the Controller in Figure 1 could use the same transaction-id for both tx-2 and tx-3 and save a single southbound transaction ID for that commit. Otherwise, the server is the one generating the ID for the transaction between the client and the server. If the client has to configure several servers, for instance to enable a network service, then each of the configured servers might return a different ID. Therefore, for a configuration modification on the client might be implemented via several southbound transactions and thus might have several southbound transaction ID.

Our proposed solution is to store, on the server, a mapping between the existing local commit id and the northbound and southbound transactions related to that local configuration change. The mapping is read only and populated by the server at configuration time as follows:

* Northbound transaction: If the set-tx-id feature is available (see Section 9.1), the server MUST accept a transaction-ID and a client ID from client supporting configuration tracing. The server MUST store both entries as respectively northbound transaction ID and

northbound client ID, associated to the local configuration ID. If the set-tx-id feature is not available, the server MUST accept the client ID, generate a transaction ID, save both the transaction ID as northbound transaction id and the client ID as northbound client ID, and send back the transaction ID to the client. If the client does not support configuration tracing, none of these entries are populated. In Figure 1, for the Controller, the northbound transaction ID is the ID of tx-1.

* Southbound transaction: If the set-tx-id feature is available (see Section 9.1), when a client has to configure servers in response to a local configuration change, then it MUST generate a transaction ID, send it along with its ID to the configured servers, and save it as a southbound transaction ID. If the set-tx-id feature is not available, it MUST sent its own ID with the configuration, receive back the transaction ID from each server, and save all of them as southbound transaction ID. In Figure 1, for the Controller, the southbound transaction IDs are the IDs of tx-2 and tx-3.

The two cases above are not mutually exclusive. A Controller can be configured by an Orchestrator and configure network equipment in turn, as shown in Figure 1. In that case, both the northbound transaction ID, shared with the Orchestrator and the southbound transaction IDs, shared with the network equipments, are stored in the Controller. They are both associated to the corresponding configuration commit in the Controller.

It is technically possible that several clients push configuration to the candidate configuration datastore and only one of them commits the changes to the running configuration datastore. From the running configuration datastore perspective, which is the effective one, there is a single modification, but caused by several clients, which means that this modification should have several northbound transaction id. Although, this case is technically possible, it is a bad practice. We won't cover it in this document. In other terms, we assume that a given configuration modification on a server is caused by a single northbound transaction, and thus has a single corresponding northbound transaction ID.

4.2. Using the YANG module

The YANG module defined below enables tracing a configuration change in a Network Equipment back to its origin, for instance a service request in an orchestrator. To do so, the Anomaly Detection System (ADS) should have for each NMS ID (as stored in northbound-client-id), access to some credentials enabling read access to the model. It should as well have access to the network equipment in which an

issue is detected.

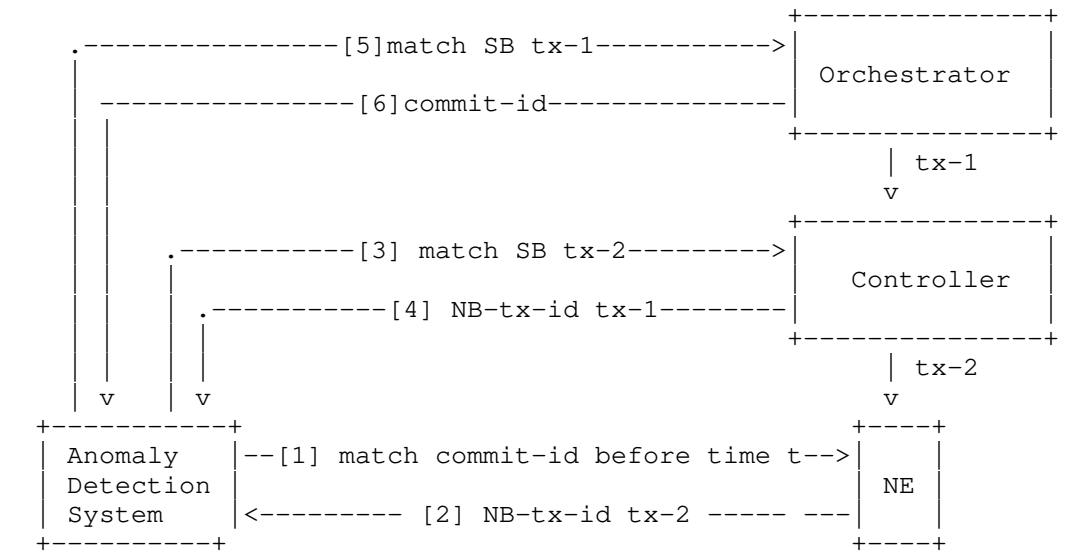

Figure 2: Example of Configuration Tracing. tx: transaction, NB: northbound, SB: southbound. The number between square brackets refer to steps in the listing below.

The steps for a software to trace a configuration modification in a Network Equipment back to a service request are illustrated in Figure 2. They are detailed below.

1. The Anomaly Detection System identifies the commit id that created an issue, for instance by looking for the last commit-id occuring before the issue was detected. The ADS queries the NE for the northbound transaction-id and northbound client id associated to the commit-id.

2. The ADS receives the northbound transaction Id. In Figure 2, that step would recieve the id of tx-2 and the id of the Controller as a result. If they are no results, or no associated northbound-transaction-id, the change was not done by a client compatible with the present draft, and the investigation stops here.

3. The ADS queries the client identified by the northbound-client-id found at the previous step, looking for a match of the northbound-transaction-id from the previous step with a southbound-transaction-id in the client version of the YANG

model. In Figure 2, for that step, the software would look for the id of tx-2 in the southbound transaction IDs stored in the Controller.

4. From that query, the ADS knows the local-commit-id on the client (Controller in our case). Since the local-commit-id is associated to a northbound-transaction-id, namely the id of tx-1, the ADS continues the investigation. The client to query is identified by the northbound-client-id, in our case the Orchestrator.

5. The ADS queries the Orchestrator, trying to find a match for the Id of tx-1 as a southbound transaction ID.

6. Finally, the ADS receives the commit-id from the Orchestrator that ultimately caused the issue in the NE. Since there is no associated northbound transaction id, the investigation stops here. The modification associated to the commit-id, for instance a service request, is now available for further manual or automated analysis, such as analyzing the root cause of the issue.

Note that step 5 and 6 are actually a repetition of step 3 and 4. The general algorithm is to continue looking for a client until no more client (no more northbound-transaction-id) can be found in the current element.

5. YANG module

We present in this section the YANG module for modelling the information about the configuration modifications.

5.1. Overview

The tree representation [RFC8340] of our YANG module is depicted in Figure 3

```
module: ietf-external-transaction-id
  +--ro external-transactions-id
     +--ro configuration-change* [local-commit-id]
        +--ro local-commit-id              string
        +--ro northbound-transaction-id?   ietf-netconf-txid:etag-t
        +--ro northbound-client-id         string
        +--ro southbound-transaction-id*   ietf-netconf-txid:etag-t
```

Figure 3: Tree representation of ietf-external-transaction-id YANG module

The local-commit-id represents the local id of the configuration changes. It can be used to retrieve the local configuration changes that happened during that transaction.

The northbound-transaction-id should be present when the server is configured by a client supporting the external transaction ID. In that case, the northbound-client-id is mandatory. The value of both fields are sent by the client whenever it sends the configuration that trigger the changes associated to the local-commit-id.

The southbound-transaction-id should be present when the current configuration change leads to the configuration of other devices. In that case, the southbound-transaction-id should be generated by the server (and unique among other southbound-transaction-id fields generated on this server), sent to the configured devices and saved in that field. If the configured server do not support having a forced transaction id, then the transaction IDs resulting of the configuration of the servers must be stored in that list.

Even if this document focuses only on NETCONF, the use cases defined in Section 3 are not specific to NETCONF and the mechanism described in this document could be adapted to other configuration mechanisms. For instance, a configuration modification pushed via CLI can be identified via a label. As such cases are difficult to standardize, we won't cover them in this document. However, our model could be extended to support such mechanism for instance by using a configuration label instead of the northbound transaction ID.

5.2. YANG module ietf-external-transaction-id

```
<CODE BEGINS> file "ietf-external-transaction-id@2021-11-03.yang"
module ietf-external-transaction-id {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-external-transaction-id";
  prefix ext-txid;

  import ietf-netconf-txid {
    prefix ietf-netconf-txid;
  }

  organization
    "IETF OPSAWG Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>
     Author:   Benoit Claise  <mailto:benoit.claise@huawei.com>
     Author:   Jean Quilbeuf  <mailto:jean.quilbeuf@huawei.com>";
```

```
  description
    "This module enable tracing of configuration changes in an
     automated network. It stores the ID of the northbound
     transaction when the local device is configured by an enabled
     NMS, and the southbound transaction ID when the local device
     configures other devices.

     The main usage of this module is to map a local configuration
     change to a northbound transaction ID that can be retrieved as
     southbound transaction ID on the configuring NMS, or to map a
     southbound transaction ID to a northbound transaction ID on
     devices that are both configured and configuring other devices.

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

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

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the Simplified BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
     This version of this YANG module is part of RFC XXXX; see the
     RFC itself for full legal notices.  ";

  revision 2021-11-03 {
    description
      "Initial revision";
    reference
      "RFC xxxx: Title to be completed";
  }

  container external-transactions-id {
    config false;
    description
      "Contains the IDs of configuration transactions that are
       external to the current device.";
    list configuration-change {
      key "local-commit-id";
      description
        "List of configuration changes, identified by their
```

```
           local-commit-id";
        leaf local-commit-id {
          type string;
          description
            "Id as saved by the server. Can be used to retrieve
             the corresponding changes using the server mechanism
             if available.";
        }
        leaf northbound-transaction-id {
          type ietf-netconf-txid:etag-t;
          description
            "External transaction ID, sent by the client, corresponding
             to a change initiated by a northbound NMS. There should be
             a corresponding entry on the NMS as a
             southbound-transaction-id that maps to the actual
             configuration commit that triggered the configuration of
             this server.

             This field is present only when the configuration was
             pushed by a compatible system.";
        }
        leaf northbound-client-id {
          when '../northbound-transaction-id';
          type string;
          mandatory true;
          description
            "ID of the client doing the modification, to further query
             information about the corresponding change.";
        }
        leaf-list southbound-transaction-id {
          type ietf-netconf-txid:etag-t;
          description
            "Transaction ID transmitted to southbound devices
             configured following the configuration change
             corresponding to local-commit-id. ";
        }
      }
    }
  }
  <CODE ENDS>
```

6. Security Considerations

7. IANA Considerations

This document includes no request to IANA.

8. Contributors

9. Open Issues / TODO

* Evaluate risk of collision between transaction ids in the southbound-transaction id. Example scenario: 1) client configures server 1 and server 2 for commit-id (client) 1 the southbound transaction IDs are A (server 1) B (server 2) 2) client configures server 1 and server 2 for commit-id (client) 2 the southbound transaction IDs are B (server 1) C (server 2) 3) the last configuration of server 1 causes an issue, when looking for southbound transaction id B, it's not clear whether the issue comes from commit 1 or commit 2 in the client

9.1. Possibility of setting the transaction Id from the client

In the -00 version of [I-D.lindblad-netconf-transaction-id], there is the possibility for the client to set the transaction id when sending the configuration to the server. This feature has been removed in subsequent versions. In this draft, we call this feature set-tx-id. Such a feature would simplify the present draft, therefore we try to present two versions, one with the feature set-tx-id available and one without.

10. Normative References

[I-D.lindblad-netconf-transaction-id]
Lindblad, J., "Transaction ID Mechanism for NETCONF", Work in Progress, Internet-Draft, draft-lindblad-netconf-transaction-id-02, 8 June 2022, <https://www.ietf.org/archive/id/draft-lindblad-netconf-transaction-id-02.txt>.

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

[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>.

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

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

11. Informative References

[I-D.ietf-opsawg-service-assurance-architecture] Claise, B., Quilbeuf, J., Lopez, D., Voyer, D., and T. Arumugam, "Service Assurance for Intent-based Networking Architecture", Work in Progress, Internet-Draft, draft-ietf-opsawg-service-assurance-architecture-11, 18 October 2022, <https://www.ietf.org/archive/id/draft-ietf-opsawg-service-assurance-architecture-11.txt>.

[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>.

Appendix A. Changes between revisions

Initial version

Appendix B. Tracing configuration changes

Acknowledgements

Authors' Addresses

Jean Quilbeuf
Huawei
Email: jean.quilbeuf@huawei.com

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

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

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 82
Madrid 28006
Spain
Email: diego.r.lopez@telefonica.com

Qiong Sun
China Telecom
Email: sunqiong@chinatelecom.cn

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 15 April 2023

R. Gagliano
Cisco Systems
K. Larsson
Deutsche Telekom AG
J. Lindblad
Cisco Systems

12 October 2022

NETCONF Extension to support Trace Context propagation
draft-rogaglia-netconf-trace-ctx-extension-00

Abstract

This document defines how to propagate trace context information across the Network Configuration Protocol (NETCONF), that enables distributed tracing scenarios. It is an adaption of the HTTP-based W3C specification.

About This Document

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

The latest revision of this draft can be found at TBD. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rogaglia-netconf-trace-ctx-extension/.

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

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

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/.

Copyright Notice

Table of Contents

1. Introduction

Network automation and management systems commonly consist of multiple sub-systems and together with the network devices they manage, they effectively form a distributed system. Distributed tracing is a methodology implemented by tracing tools to follow, analyze and debug operations, such as configuration transactions, across multiple distributed systems. An operation is uniquely identified by a trace-id and through a trace context, carries some metadata about the operation. Propagating this "trace context" between systems enables forming a coherent view of the entire operation as carried out by all involved systems.

The W3C has defined two HTTP headers for context propagation that are useful in use case scenarios of distributed systems like the ones defined in [RFC8309]. This document defines an extension to the NETCONF protocol to add the same concepts and enable trace context propagation over NETCONF.

It is worth noting that the trace context is not meant to have any relationship with the data that is carried with a given operation (including configurations, service identifiers or state information).

A trace context also differs from [I-D.lindblad-netconf-transaction-id] in several ways as the trace operation may involve any operation (including for example validate, lock, unlock, etc.) Additionally, a trace context scope may include the full application stack (orchestrator, controller, devices, etc) rather than a single NETCONF server, which is the scope for the transaction-id.

The following enhancement of the reference SDN Architecture from RFC 8309 shows the impact of distributed traces for a network operator.

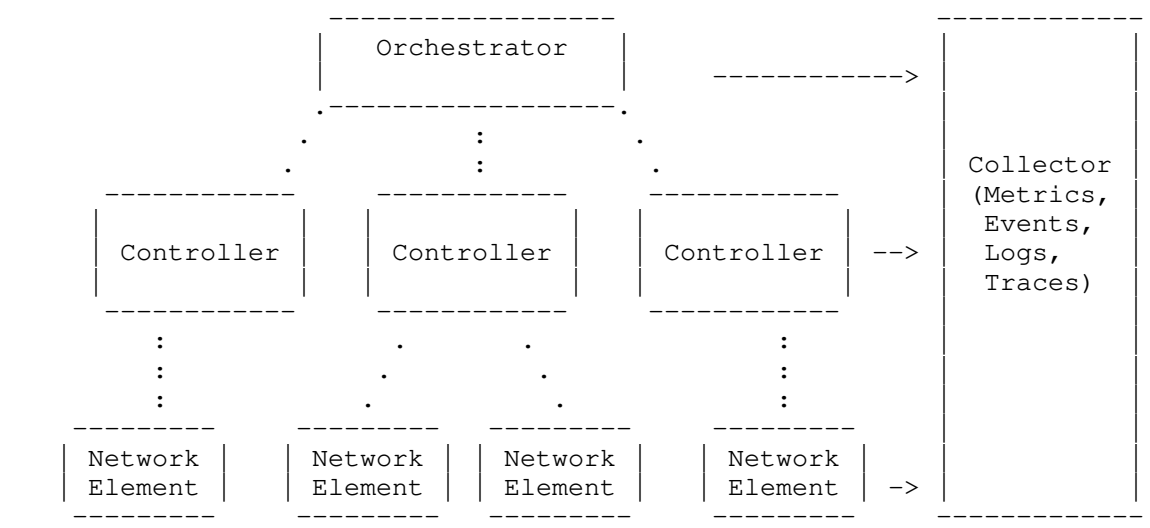

Figure 1: A Sample SDN Architecture from RFC8309 augmented to include the export of metrics, events, logs and traces from the different components to a common collector.

The network automation, management and control architectures are distributed in nature. In order to "manage the managers", operators would like to use the same techniques as any other distributed systems in their IT environment. Solutions for analysing Metrics, Events, Logs and Traces (M.E.L.T) are key for the successful monitoring and troubleshooting of such applications. Initiatives such as the OpenTelemetry [OpenTelemetry] enable rich ecosystems of tools that NETCONF-based applications would want to participate in.

With the implementation of this trace context propagation extension to NETCONF, backend systems behind the M.E.L.T collector will be able to correlate information from different systems but related to a common context.

1.1. Implementation example: OpenTelemetry

We will describe an example to show the value of trace context propagation in the NETCONF protocol. In Figure 2, we show a deployment based on Figure 1 with a single controller and two network elements. In this example, the NETCONF protocol is running between the Orchestrator and the Controller. NETCONF is also used between the Controller and the Network Elements.

Let's assume an edit-config operation between the orchestrator and the controller that results (either synchronously or asynchronously) in corresponding edit-config operations from the Controller towards the two network elements. All trace operations are related and will create M.E.L.T data.

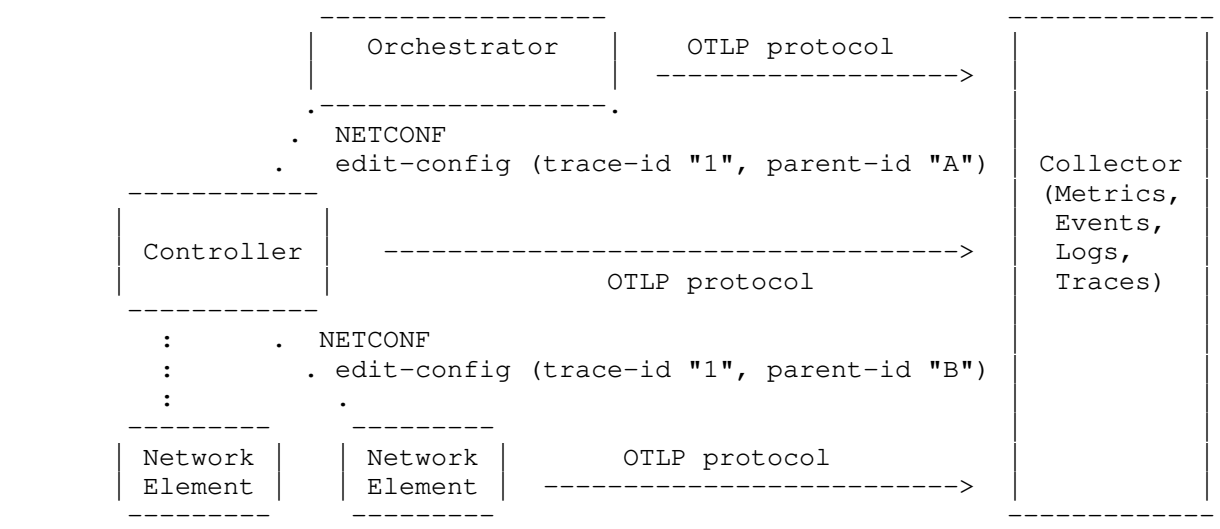

Figure 2: An implementation example where the NETCONF protocol is used between the Orchestrator and the Controller and also between the Controller and the Network Elements. Every component exports M.E.L.T information to the collector using the OTLP protocol.

Each of the components in this example (Orchestrator, Controller and Network Elements) is exporting M.E.L.T information to the collector using the OpenTelemetry Protocol (OTLP).

For every edit-config operation, the trace context is included. In particular, the same trace-id "1" (simplified encoding for documentation) is included in all related NETCONF messages, which enables the collector and any backend application to correlate all M.E.L.T messages related to this transaction in this distributed stack.

Another interesting attribute is the parent-id. We can see in this example that the parent-id between the orchestrator and the controller ("A") is different from the one between the controller and the network elements ("B"). This attribute will help the collector and the backend applications to build a connectivity graph to understand how M.E.L.T information exported from one component relates to the information exported from a different component.

With this additional metadata exchanged between the components and exposed to the M.E.L.T collector, there are important improvements to the monitor and troubleshooting operations for the full application stack.

1.2. Use Cases

1.2.1. Provisioning root cause analysis

When a provisioning activity fails, errors are typically propagated northbound, however this information may be difficult to troubleshoot and typically, operators are required to navigate logs across all the different components.

With the support for trace context propagation as described in this document for NETCONF, the telemetry collector will be able to search every trace, event, metric, or log in connection to that trace-id and perform a root cause analysis.

1.2.2. System performance profiling

When operating a distributed system such as the one shown in Figure 2, operators are expected to benchmark what are the Key Performance Indicators (KPIs) for the most common tasks. For example, what is the typical delay when provisioning a VPN service across different controllers and devices.

Thanks to Application Performance Management (APM) systems, from these KPIs, an operator can detect a normal and abnormal behaviour of the distributed system. Also, an operator can better plan any upgrades or enhancements in the platform.

With the support for context propagation as described in this document for NETCONF, much richer system-wide KPIs can be defined and used for troubleshooting as the metrics and traces propagated by the different components share a common context. Troubleshooting for abnormal behaviours can also be troubleshot from the system view down to the individual element.

1.3. 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 XML prefixes used in this document are mapped as follows:

* xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0",

* xmlns:notif="urn:ietf:params:xml:ns:netconf:notification:1.0",

* xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-patch" and

* xmlns:ypatch="urn:ietf:params:xml:ns:yang:ietf-yang-patch".

2. NETCONF Extension

When performing NETCONF operations by sending NETCONF RPCs, a NETCONF client MAY include trace context information in the form of XML attributes. The [W3C-Trace-Context] defines two HTTP headers; traceparent and tracestate for this purpose. NETCONF clients that are taking advantage of this feature MUST add one w3ctc:traceparent attribute to the nc:rpc tag.

A NETCONF server that receives a trace context attribute in the form of a w3ctc:traceparent attribute SHOULD apply the mutation rules described in [W3C-Trace-Context]. A NETCONF server MAY add one w3ctc:traceparent attribute in the nc:rpc-reply response to the nc:rpc tag above. NETCONF servers MAY also add one w3ctc:traceparent attribute in notification and update message envelopes: notif:notification, yp:push-update and yp:push-change-update.

For example, a NETCONF client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1"
     xmlns:w3ctc="urn:ietf:params:xml:ns:netconf:w3ctc:1.0"
     w3ctc:traceparent=
       "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01">
  <get-config/>
</rpc>
```

In all cases above where a client or server adds a w3ctc:traceparent attribute to a tag, that client or server MAY also add one w3ctc:tracestate attribute to the same tag.

The proper encoding and interpretation of the contents of the w3ctc:traceparent attribute is described in [W3C-Trace-Context] section 3.2 except 3.2.1. The proper encoding and interpretation of the contents in the w3ctc:tracestate attribute is described in [W3C-Trace-Context] section 3.3 except 3.3.1 and 3.3.1.1. A NETCONF tag can only have zero or one w3ctc:tracestate attributes, so its content MUST always be encoded as a single string. The tracestate field value is a list of list-members separated by commas (,). A list-member is a key/value pair separated by an equals sign (=). Spaces and horizontal tabs surrounding list-members are ignored. There is no limit to the number of list-members in a list.

For example, a NETCONF client might send:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1"
     xmlns:w3ctc="urn:ietf:params:xml:ns:netconf:w3ctc:1.0"
     w3ctc:tracestate="rojo=00f067aa0ba902b7,congo=t61rcWkgMzE"
     w3ctc:traceparent=
       "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01">
  <get-config/>
</rpc>
```

As in all XML documents, the order between the attributes in an XML tag has no significance. Clients and servers MUST be prepared to handle the attributes no matter in which order they appear. The tracestate value MAY contain double quotes in its payload. If so, they MUST be encoded according to XML rules, for example:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1"
     xmlns:w3ctc="urn:ietf:params:xml:ns:netconf:w3ctc:1.0"
     w3ctc:traceparent=
       "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01"
     w3ctc:tracestate=
       "value-with-quotes=&quot;Quoted string&quot;,other-value=123">
  <get-config/>
</rpc>
```

TBD Errors ....

3. Security Considerations

TODO Security

4. IANA Considerations

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

urn:ietf:params:netconf:capability:w3ctc:1.0

This document registers one XML namespace URN in the 'IETF XML registry', following the format defined in [RFC3688] (https://tools.ietf.org/html/rfc3688).

URI: urn:ietf:params:xml:ns:netconf:w3ctc:1.0

Registrant Contact: The NETCONF WG of the IETF.

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

5. Acknowledgments

TBD

6. References

6.1. Normative References

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

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

[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>.

[W3C-Trace-Context] "W3C Recommendation on Trace Context", 23 November 2021, <https://www.w3.org/TR/2021/REC-trace-context-1-20211123/>.

6.2. Informative References

[I-D.lindblad-netconf-transaction-id] Lindblad, J., "Transaction ID Mechanism for NETCONF", Work in Progress, Internet-Draft, draft-lindblad-netconf-transaction-id-02, 8 June 2022, <https://www.ietf.org/archive/id/draft-lindblad-netconf-transaction-id-02.txt>.

[OpenTelemetry]
"OpenTelemetry Cloud Native Computing Foundation project", 29 August 2022, <https://opentelemetry.io>.

[RFC8309] Wu, Q., Liu, W., and A. Farrel, "Service Models Explained", RFC 8309, DOI 10.17487/RFC8309, January 2018, <https://www.rfc-editor.org/info/rfc8309>.

[W3C-Baggage]
"W3C Propagation format for distributed context Baggage", 23 November 2021, <https://www.w3.org/TR/baggage/#examples-of-http-headers>.

Appendix A. TO DO List (to be deleted by RFC Editor)

* Manage versioning of the trace-context specification

* We intend to extend the trace-concext capability to RESTCONF in a future draft

* The W3C is working on a draft document to introduce the concept of "baggage" [W3C-Baggage] that we expect part of a future draft for NETCONF and RESTCONF

Appendix B. XML Attributes vs RPCs input augmentations discussion (to be deleted by RFC Editor)

There are arguments that can be raised regarding using XML Attribute or to augment NETCONF RPCs.

We studied Pros/Cons of each option and decided to propose XML attributes:

XML Attributes Pro:

* Literal alignment with W3C specification

* Same encoding for RESTCONF and NETCONF enabling code reuse

* One specification for all current and future rpcs

XML Attributes Cons:

* No YANG modeling, multiple values represented as a single string

* Dependency on W3C for any extension or changes in the future as encoding will be dictated by string encoding

RPCs Input Augmentations Pro:

* YANG model of every leaf
* Re-use of YANG toolkits
* Simple updates by augmentations on existing YANG module
* Possibility to express deviations in case of partial support

RPCs Input Augmentations Cons:

* Need to augment every rpc, including future rpcs would need to consider these augmentations, which is harder to maintain
* There is no literal alignment with W3C standard. However, as mentioned before most of the time there will be modifications to the content
* Would need updated RFP for each change at W3C, which will make adoption of new features slower

Authors' Addresses

Roque Gagliano
Cisco Systems
Avenue des Uttins 5
CH-1180 Rolle
Switzerland
Email: rogaglia@cisco.com

Kristian Larsson
Deutsche Telekom AG
Email: kll@dev.terastrm.net

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2023

T. Graf
Swisscom
B. Claise
Huawei
A. Huang Feng
INSA-Lyon
20 October 2022

Support of Versioning in YANG Notifications Subscription
draft-tgraf-netconf-yang-notifications-versioning-00

Abstract

This document extends the YANG notifications subscription mechanism to specify the YANG module semantic version at the subscription. Then, a new extension with new metadata of the YANG push update notification header is proposed.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 23 April 2023.

Copyright Notice

Table of Contents

1. Introduction

In order to process the received YANG push notification messages described in section 3.7 of [RFC8641] at the YANG push receiver, a semantic reference to the YANG module and the XPath or subtree is needed to determine the data types for each field and which part of the YANG module the metrics are expose from.

This specification applies to the YANG push configured subscriptions defined in Section 2.5 of [RFC8639], where a publisher is configured to stream notification out of band, as opposed to dynamic subscriptions defined in Section 2.4 of [RFC8641], where the subscriber can initiate and modify the subscription dynamically in-band. In the latter case, the subscriber knows already all the subscriber YANG-related information, which it has to know in order to configure the subscription.

This semantic reference is available during the subscription period described in Section 3.6 of [RFC8641] where for each subscription a locally unique subscription ID described in Section 4.3.2 of [RFC8641] is being issued and streamed as metadata with the notification message in the YANG push message header. This implies that the YANG receiver needs to lookup the subscription inventory for a subscription ID in each notification message to determine the YANG module name, namespace, and filter definitions.

The semantics can change between different YANG module revisions. The YANG module version statement is specified in Section 7.1.2 of [RFC6020] and states that the newer revision needs to be backward compatible to the previous revision. Section 3.1 of [I-D.ietf-netmod-yang-module-versioning] specifies that newer semantic versions introduced in [I-D.ietf-netmod-yang-semver] MAY not be backward compatible to the previous version when indicated with non-backwards-compatible keyword.

The YANG notifications subscription mechanism defined in [RFC8641] does not allow to specify the YANG module revision. When a network node is upgraded, the subscribed YANG module revision MAY have updated and might, consequently, break the data processing pipeline since the YANG push receiver may not be aware of this change.

This documents extends the current YANG notifications subscription mechanism to allow to subscribe to a specific revision or latest semantic version to which the YANG module version needs to be backward compatible to and advertises its specific YANG module revision inband. Besides the existing Subscription ID, the YANG module name, namespace, revision and filtering metadata is added to the YANG push notification header to lift the YANG push receiver need to lookup the subscription inventory information.

2. Extend the Datastore Selection

The YANG notifications subscription OPTIONALLY can be restricted to the following YANG module revision for future capabilities:

latest: Restricts the subscription to the latest YANG module revision.

revision: Restricts the subscription to a specific YANG module revision. Example 2014-05-08.

latest-compatible-semversion: Restricts the subscription to the latest compatible YANG module semantic version referenced to. Example 2.0.0.

If nothing is specified, latest YANG module version is implied.

3. Extend the Metadata in Streaming Update

Along with the subscribed content, besides the Subcription ID, the following metadata objects are part of a "push-update" or "push-change-update" notification.

module: Describes the YANG module name for the related streamed content.

namespace: Describes the YANG module namespace as specified in Section 7.1.3 of [RFC6020] for the related streamed content.

revision: Describes the YANG module revision as specified in Section 7.1.2 of [RFC6020] for the related streamed content.

revision-label: Describes the YANG module semantic version as specified in [I-D.ietf-netmod-yang-semver] for the related streamed content.

datastore-xpath-filter: Describes the YANG module xpath filter as specified in Section 6.4 of [RFC6020] for the related streamed content.

datastore-subtree-filter: Describes the YANG module subtree filter as specified in Section 6 of [RFC6241] for the related streamed content.

Figure 1 provides an example of a notification message with the YANG module name, revision, revision label and datastore-xpath-filter for a subscription tracking the operational status of a single Ethernet interface (per [RFC8343]). This notification message is encoded XML [W3C.REC-xml-20081126] over the Network Configuration Protocol (NETCONF) as per [RFC8640].

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2022-09-02T10:59:55.32Z</eventTime>
 <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <id>101</id>
   <module>ietf-interfaces</module>
   <namespace>urn:ietf:params:xml:ns:yang:ietf-interfaces</namespace>
   <revision>2014-05-08</revision>
   <revision-label>1.0.0</revision-label>
   <datastore-xpath-filter>ietf-interfaces:interfaces</datastore-xpath-filter>
   <datastore-contents>
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
       <interface>
         <name>eth0</name>
         <oper-status>up</oper-status>
       </interface>
     </interfaces>
   </datastore-contents>
 </push-update>
</notification>
```

Figure 1: XML Push Example for a Periodical Notification Message

Figure 2 provides an example of a JSON encoded, [RFC8259], notification message over HTTPS-based [I-D.ietf-netconf-https-notif] or UDP-based [I-D.ietf-netconf-udp-notif] transport for the same subscription.

```
{
    "ietf-notification:notification": {
        "eventTime": "2022-09-02T10:59:55.32Z",
        "ietf-yang-push:push-update": {
            "id": 101,{
            "module": "ietf-interfaces",{
            "namespace": "urn:ietf:params:xml:ns:yang:ietf-interfaces",{
            "revision": "2014-05-08",{
            "revision-label": "1.0.0",{
            "datastore-xpath-filter": "ietf-interfaces:interfaces",
            "datastore-contents": {
                "ietf-interfaces:interface": {
                    "name": {
                        "eth0": {
                                "oper-status": "up",
                                }
                            }
                        }
                    }
                }
            }
        }
   }
}
```

Figure 2: JSON Push Example for a Periodical Notification Message

4. The "ietf-yang-push-metadata" Module

4.1. Data Model Overview

This YANG module augments the "ietf-yang-push" module to add subcription metadata into the "push-update" and "push-change-update" notification and the ability to define the "revision" and "revision-label" in the "establish-subscription" and "modify-subscription" in the datastore push subscription.

```
module: ietf-yang-push-metadata

  augment /yp:push-update:
    +--ro module?                     string
    +--ro namespace?                  string
    +--ro revision?                   rev:revision-date-or-label
    +--ro revision-label?             ysver:version
    +--ro datastore-xpath-filter?     yang:xpath1.0 {sn:xpath}?
    +--ro datastore-subtree-filter?   <anydata> {sn:subtree}?
  augment /yp:push-change-update:
    +--ro module?                     string
    +--ro namespace?                  string
    +--ro revision?                   rev:revision-date-or-label
    +--ro revision-label?             ysver:version
    +--ro datastore-xpath-filter?     yang:xpath1.0 {sn:xpath}?
    +--ro datastore-subtree-filter?   <anydata> {sn:subtree}?
  augment /sn:establish-subscription/sn:input/sn:target:
    +-- revision?         rev:revision-date-or-label
    +-- revision-label?   ysver:version
  augment /sn:modify-subscription/sn:input/sn:target:
    +-- revision?         rev:revision-date-or-label
    +-- revision-label?   ysver:version
  augment /sn:subscription-started/sn:target:
    +-- revision?         rev:revision-date-or-label
    +-- revision-label?   ysver:version
  augment /sn:subscription-modified/sn:target:
    +-- revision?         rev:revision-date-or-label
    +-- revision-label?   ysver:version
  augment /sn:subscriptions/sn:subscription/sn:target:
    +--rw revision?         rev:revision-date-or-label
    +--rw revision-label?   ysver:version
```

4.2. YANG Module

The YANG module has six leaves augmenting the model of YANG-push [RFC8641].

```
<CODE BEGINS> file "ietf-yang-push-metadata@2022-10-10.yang"
module ietf-yang-push-metadata {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-push-metadata";
  prefix ypm;
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
```

```
  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-yang-push {
    prefix yp;
    reference
      "RFC 8641: Subscription to YANG Notifications for Datastore Updates";
  }
  import ietf-yang-revisions {
    prefix rev;
    reference
      "RFC XXXX: draft-ietf-netmod-yang-module-versioning-06, Updated YANG
      Module Revision Handling";
  }
  import ietf-yang-semver {
    prefix ysver;
    reference
      "RFC XXXX: draft-ietf-netmod-yang-semver-07, YANG Semantic Versioning";
  }

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

     Authors:  Thomas Graf
               <mailto:thomas.graf@swisscom.com>
               Benoit Claise
               <mailto:benoit.claise@huawei.com>
               Alex Huang Feng
               <mailto:alex.huang-feng@insa-lyon.fr>";

  description
    "Defines YANG push event notification header with metadata.

    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; see the RFC
    itself for full legal notices.";
```

```
     revision 2022-09-21 {
       description
         "First revision";
       reference
         "RFC XXXX: Support of Versioning in YANG Notifications Subscription";
     }

     feature yang-push-metadata-supported {
       description
         "This feature indicates the YANG Notifications have the metadata
         defined in this YANG module.";
     }

     grouping yang-push-module-version {
       leaf revision {
         type rev:revision-date-or-label;
         description
           "This references the YANG module revision of the sent notification.";
       }
       leaf revision-label {
         type ysver:version;
         description
           "This references the YANG module semversion of the sent notification."
;
       }
     }

     grouping yang-push-metadata {
       leaf module {
         type string;
         description
           "This references the YANG module of the sent notification.";
       }
       leaf namespace {
         type string;
         description
           "This references the YANG module namespace of the sent notification.";
       }

       uses ypm:yang-push-module-version;

       leaf datastore-xpath-filter {
         type yang:xpath1.0;
         if-feature "sn:xpath";
         description
           "This references the YANG module xpath of the sent notification.";
       }
       anydata datastore-subtree-filter {
         if-feature "sn:subtree";
```

```
          description
            "This references the YANG module subtree of the sent notification.";
        }
      }
      // Event notifications
      augment "/yp:push-update" {
        description
          "This augmentation adds the module, the namespace, the revision, the
          semversion, the xpath and the subtree in the push-update notification";
        uses ypm:yang-push-metadata;
      }

      augment "/yp:push-change-update" {
        description
          "This augmentation adds the module, the namespace, the revision, the
          semversion, the xpath and the subtree in the push-change-update notifica
tion";
          uses ypm:yang-push-metadata;
      }

      // Subscription parameters
      augment "/sn:establish-subscription/sn:input/sn:target" {
        uses ypm:yang-push-module-version;
      }
      augment "/sn:modify-subscription/sn:input/sn:target" {
        uses ypm:yang-push-module-version;
      }

      // Subscription notifications
      augment "/sn:subscription-started/sn:target" {
        uses ypm:yang-push-module-version;
      }
      augment "/sn:subscription-modified/sn:target" {
        uses ypm:yang-push-module-version;
      }

      // Subscription container
      augment "/sn:subscriptions/sn:subscription/sn:target" {
        uses ypm:yang-push-module-version;
      }
    }
    <CODE ENDS>
```

5. Security Considerations

The security considerations for the YANG notifications subscription mechanism are described in [RFC8641]. This documents adds no additional security considerations.

6. IANA Considerations

This document has no IANA actions.

7. Acknowledgements

The authors would like to thank xxx for their review and valuable comments.

8. References

8.1. Normative References

[I-D.ietf-netmod-yang-module-versioning]
Wilton, R., Rahman, R., Lengyel, B., Clarke, J., and J. Sterne, "Updated YANG Module Revision Handling", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-versioning-06, 10 July 2022, <https://www.ietf.org/archive/id/draft-ietf-netmod-yang-module-versioning-06.txt>.

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

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

[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>.

[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>.

[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>.

8.2. 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-12, 22 August 2022, <https://www.ietf.org/archive/id/draft-ietf-netconf-https-notif-12.txt>.

[I-D.ietf-netconf-udp-notif]
Zheng, G., Zhou, T., Graf, T., Francois, P., Feng, A. H., and P. Lucente, "UDP-based Transport for Configured Subscriptions", Work in Progress, Internet-Draft, draft-ietf-netconf-udp-notif-08, 12 September 2022, <https://www.ietf.org/archive/id/draft-ietf-netconf-udp-notif-08.txt>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

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

[RFC8640] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Dynamic Subscription to YANG Events and Datastores over NETCONF", RFC 8640, DOI 10.17487/RFC8640, September 2019, <https://www.rfc-editor.org/info/rfc8640>.

[W3C.REC-xml-20081126]
Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <https://www.w3.org/TR/2008/REC-xml-20081126>.

Authors' Addresses

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

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

Alex Huang Feng
INSA-Lyon
Lyon
France
Email: alex.huang-feng@insa-lyon.fr