Network Working Group
Internet-Draft
Updates: RFC5277 (if approved)
Intended status: Standards Track
Expires: 22 April 2024

A. Huang Feng
P. Francois
INSA-Lyon
T. Graf
Swisscom
B. Claise
Huawei

20 October 2023

YANG model for NETCONF Event Notifications
draft-ahuang-netconf-notif-yang-03

Abstract

This document defines the YANG model for NETCONF Event Notifications. The definition of this YANG model allows the encoding of NETCONF Event Notifications in YANG compatible encodings such as YANG-JSON and YANG-CBOR.

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 22 April 2024.

Table of Contents

1. Introduction

This document defines a YANG [RFC7950] data model for NETCONF Event Notifications [RFC5277]. The notification structure defined in [RFC5277] uses a XML Schema [W3C.REC-xml-20001006] allowing to encode and validate the message in XML. Nevertheless, when the notification message is encoded using other encodings such as YANG-JSON [RFC7951] or YANG-CBOR [RFC9254], a YANG model to validate or encode the message is necessary. This document extends [RFC5277], defining the NETCONF Event Notification structure in a YANG module.

2. Differences to draft-ietf-netconf-notification-messages

[I-D.ietf-netconf-notification-messages] proposes a structure to send multiple notifications in a single message. Unlike [I-D.ietf-netconf-notification-messages], this document defines a YANG module to encode NETCONF Notifications with encodings other than XML, which is currently not existing. The structure for NETCONF notifications is defined in [RFC5277] using a XSD, but there is no YANG module defining the structure of the notification message sent by a server when the message is encoded in YANG-JSON [RFC7951] or YANG-CBOR [RFC9254].

3. YANG Module

3.1. YANG Tree Diagram

This YANG module adds a structure with one leaf for the datetime as defined in section 2.2.1 of [RFC5277]. The name of the leaf matches the definition of the XSD element name defined in Section 4 of [RFC5277].

```
module: ietf-notification

  structure notification:
    +-- eventTime    yang:date-and-time
```

3.2. YANG Module

The YANG module uses the same namespace from the XML Schema defined in Section 4 of [RFC5277] allowing to use this YANG module to also validate already implemented XML encoded NETCONF Event Notifications.

```
<CODE BEGINS> file "ietf-notification@2023-07-23.yang"
module ietf-notification {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:netconf:notification:1.0";
  prefix inotif;
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions";
  }
```

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

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

     description
       "Defines NETCONF Event Notification structure as defined in RFC5277.
       This YANG module uses the same namespace from the XML schema defined
       in Section 4 of RFC5277 to be able to validate already implemented
       XML encoded messages.

       Copyright (c) 2023 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

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

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

     revision 2023-07-23 {
       description
         "First revision";
       reference
         "RFC XXXX: NETCONF Event Notification YANG";
     }

     sx:structure notification {
       leaf eventTime {
         type yang:date-and-time;
         mandatory true;
         description
           "The date and time the event was generated by the event source.
           This parameter is of type dateTime and compliant to [RFC3339].
           Implementations must support time zones.
           The leaf name in camel case matches the name of the XSD element
```

```
            defined in Section 4 of RFC5277.";
        }
      }
    }
    <CODE ENDS>
```

4. Security Considerations

The security considerations for the NETCONF Event notifications are described in [RFC5277]. This documents adds no additional security considerations.

5. IANA Considerations

This document describes the URI used for the IETF XML Registry and registers a new YANG module name.

5.1. URI

IANA is requested to add this document as a reference in the following URI in the IETF XML Registry [RFC3688].

URI: urn:ietf:params:xml:ns:netconf:notification:1.0
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.
Reference: RFC5277; RFC-to-be

5.2. YANG module name

This document registers the following YANG module in the YANG Module Names Registry [RFC6020], within the "YANG Parameters" registry:

name: ietf-notification
namespace: urn:ietf:params:xml:ns:netconf:notification:1.0
prefix: inotif
reference: RFC-to-be

6. Acknowledgements

The authors would like to thank Andy Bierman, Tom Petch and Jason Sterne for their review and valuable comments.

7. References

7.1. 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>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

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

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

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

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

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

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

[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>.

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

[W3C.REC-xml-20001006] Bray, T., Paoli, J., Sperberg-McQueen, M., and E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C, October 2000, <https://www.w3.org/TR/2000/REC-xml-20001006>.

7.2. Informative References

[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://datatracker.ietf.org/doc/html/draft-ietf-netconf-notification-messages-08>.

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

Appendix A. Examples

This non-normative section shows an example of how a YANG-JSON and YANG-CBOR are encoded.

A.1. YANG-JSON encoded message

This is an example of a YANG-JSON encoded notification.

```
{
    "ietf-notification:notification": {
        "eventTime": "2023-02-10T08:00:11.22Z",
        "ietf-yang-push:push-update": {
            "id": 1011,
            "datastore-contents": {
                "ietf-interfaces:interfaces": [
                    {
                        "interface": {
                            "name": "eth0",
                            "oper-status": "up"
                        }
                    }
                ]
            }
        }
    }
}
```

Figure 1: JSON-encoded notification

A.2. YANG-CBOR encoded message

This is an example of YANG-CBOR encoded notification. The figure Figure 2 shows the message using the CBOR diagnostic notation as defined in section 3.1 of [RFC9254].

```
{
    "ietf-notification:notification": {
        "eventTime": "2023-02-10T08:00:11.22Z",
        "ietf-yang-push:push-update": {
            "id": 1011,
            "datastore-contents": {
                "ietf-interfaces:interfaces": [
                    {
                        "interface": {
                            "name": "eth0",
                            "oper-status": "up"
                        }
                    }
                ]
            }
        }
    }
}
```

Figure 2: CBOR-encoded notification using diagnostic notation

Authors' Addresses

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

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

Thomas Graf
Swisscom
Binzring 17
CH-8045 Zurich
Switzerland

Email: thomas.graf@swisscom.com

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 April 2024

A. Huang Feng
P. Francois
INSA-Lyon
K. Watsen
Watsen Networks
23 October 2023

YANG Grouping for UDP Clients and UDP Servers
draft-ahuang-netconf-udp-client-server-00

Abstract

This document defines two YANG 1.1 modules to support the configuration of UDP clients and UDP servers. The modules include basic parameters for configuring UDP based clients and servers and a DTLS container when encryption needs to be enabled.

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 25 April 2024.

Copyright Notice

Table of Contents

1. Introduction

This documents defines two YANG 1.1 [RFC7950] modules to support the configuration of UDP clients and UDP servers, either as standalone or in conjunction with configuration of other protocol layers.

2. The "ietf-udp-client" Module

The "ietf-udp-client" YANG module defines two groupings for configuring UDP clients: the "udp-client-grouping" for UDP clients and the "udp-dtls-client-grouping" for UDP clients with DTLS 1.3 [RFC9147] enryption.

2.1. The "udp-client-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "udp-client-grouping" grouping:

```
   module: ietf-udp-client

     grouping udp-client-grouping:
       +-- remote-address    inet:ip-address-no-zone
       +-- remote-port       inet:port-number
```

2.2. The "udp-dtls-client-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "udp-dtls-client-grouping" grouping:

```
module: ietf-udp-client

  grouping udp-dtls-client-grouping:
    +-- remote-address    inet:ip-address-no-zone
    +-- remote-port       inet:port-number
    +-- dtls! {dtls13}?
       +-- client-identity!
       |  +-- (auth-type)
       |     +--:(certificate) {client-ident-x509-cert}?
       |     |  +-- certificate
       |     |     +-- (local-or-keystore)
       |     |           ...
       |     +--:(raw-public-key) {client-ident-raw-public-key}?
       |     |  +-- raw-private-key
       |     |     +-- (local-or-keystore)
       |     |           ...
       |     +--:(tls12-psk)
       |     |        {client-ident-tls12-psk,not tlsc:client-ident-tls12-psk}?
       |     |  +-- tls12-psk
       |     |     +-- (local-or-keystore)
       |     |     |     ...
       |     |     +-- id?                          string
       |     +--:(tls13-epsk) {client-ident-tls13-epsk}?
       |        +-- tls13-epsk
       |           +-- (local-or-keystore)
       |           |     ...
       |           +-- external-identity          string
       |           +-- hash
       |           |       tlscmn:epsk-supported-hash
       |           +-- context?                   string
       |           +-- target-protocol?           uint16
       |           +-- target-kdf?                uint16
       +-- server-authentication
       |  +-- ca-certs! {server-auth-x509-cert}?
       |  |  +-- (local-or-truststore)
       |  |     +--:(local) {local-definitions-supported}?
       |  |     |  +-- local-definition
```

```
        |  |     |          ...
        |  |     +--:(truststore)
        |  |              {central-truststore-supported,certificates}?
        |  |        +-- truststore-reference?   ts:certificate-bag-ref
        |  +-- ee-certs! {server-auth-x509-cert}?
        |  |  +-- (local-or-truststore)
        |  |     +--:(local) {local-definitions-supported}?
        |  |     |  +-- local-definition
        |  |     |          ...
        |  |     +--:(truststore)
        |  |              {central-truststore-supported,certificates}?
        |  |        +-- truststore-reference?   ts:certificate-bag-ref
        |  +-- raw-public-keys! {server-auth-raw-public-key}?
        |  |  +-- (local-or-truststore)
        |  |     +--:(local) {local-definitions-supported}?
        |  |     |  +-- local-definition
        |  |     |          ...
        |  |     +--:(truststore)
        |  |              {central-truststore-supported,public-keys}?
        |  |        +-- truststore-reference?   ts:public-key-bag-ref
        |  +-- tls12-psks?        empty
        |  |       {server-auth-tls12-psk,not tlsc:server-auth-tls12-psk}?
        |  +-- tls13-epsks?       empty {server-auth-tls13-epsk}?
        +-- hello-params {tlscmn:hello-params}?
        |  +-- tls-versions
        |  |  +-- tls-version*   identityref
        |  +-- cipher-suites
        |     +-- cipher-suite*   identityref
        +-- keepalives {tls-client-keepalives}?
           +-- peer-allowed-to-send?   empty
           +-- test-peer-aliveness!
              +-- max-wait?       uint16
              +-- max-attempts?   uint8
```

2.3. YANG Module

The "ietf-udp-client" YANG module uses the groupings defined in [I-D.ietf-netconf-tls-client-server] for configuring the DTLS 1.3 encryption.

<CODE BEGINS> file "ietf-udp-client@2023-10-16.yang"

```
module ietf-udp-client {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-udp-client";
  prefix udpc;
  import ietf-inet-types {
    prefix inet;
```

```
      reference
        "RFC 6991: Common YANG Data Types";
    }
    import ietf-tls-client {
      prefix tlsc;
      reference
        "RFC TTTT: YANG Groupings for TLS Clients and TLS Servers";
    }

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

       Authors:  Alex Huang Feng
                 <mailto:alex.huang-feng@insa-lyon.fr>
                 Pierre Francois
                 <mailto:pierre.francois@insa-lyon.fr>";

    description
      "Defines a generic grouping for UDP-based client applications.
      Supports groupings for UDP clients and UDP clients with DTLS encryption.

      Copyright (c) 2023 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

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

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

    revision 2023-10-16 {
      description
        "Initial revision";
      reference
        "RFC-to-be: YANG Grouping for UDP Clients and UDP Servers";
    }

   /*
    * FEATURES
    */
    feature dtls13 {
      description
        "This feature indicates that DTLS 1.3 encryption of UDP
```

```
          packets is supported.";
      }

      grouping udp-client-grouping {
        description
          "Provides a reusable grouping for configuring a UDP client.";

        leaf remote-address {
          type inet:ip-address-no-zone;
          mandatory true;
          description
            "IP address of the UDP client, which can be an
            IPv4 address or an IPV6 address.";
        }

        leaf remote-port {
          type inet:port-number;
          mandatory true;
          description
            "Port number of the UDP client.";
        }
      }

      grouping udp-dtls-client-grouping {
        description
          "Provides a reusable grouping for configuring a UDP client with
          DTLS encryption.";

        uses udp-client-grouping;
        container dtls {
          if-feature dtls13;
          presence dtls;
          uses tlsc:tls-client-grouping {
            // Using tls-client-grouping without TLS1.2 parameters
            // allowing only DTLS 1.3
            refine "client-identity/auth-type/tls12-psk" {
              // create the logical impossibility of enabling TLS1.2
              if-feature "not tlsc:client-ident-tls12-psk";
            }
            refine "server-authentication/tls12-psks" {
              // create the logical impossibility of enabling TLS1.2
              if-feature "not tlsc:server-auth-tls12-psk";
            }
          }
          description
            "Container for configuring DTLS 1.3 parameters.";
        }
      }
```

```
}
<CODE ENDS>
```

3. The "ietf-udp-server" Module

The "ietf-udp-server" YANG module defines two groupings for configuring UDP servers: the "udp-server-grouping" for UDP servers and the "udp-dtls-server-grouping" for UDP servers with DTLS 1.3 [RFC9147] enryption.

3.1. The "udp-server-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "udp-server-grouping" grouping:

```
module: ietf-udp-server

  grouping udp-server-grouping:
    +-- local-address    inet:ip-address-no-zone
    +-- local-port       inet:port-number
```

3.2. The "udp-dtls-server-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "udp-dtls-server-grouping" grouping:

```
module: ietf-udp-server

  grouping udp-dtls-server-grouping:
    +-- local-address    inet:ip-address-no-zone
    +-- local-port       inet:port-number
    +-- dtls! {dtls13}?
       +-- server-identity
       |  +-- (auth-type)
       |     +--:(certificate) {server-ident-x509-cert}?
       |     |  +-- certificate
       |     |     +-- (local-or-keystore)
       |     |        ...
       |     +--:(raw-private-key) {server-ident-raw-public-key}?
       |     |  +-- raw-private-key
       |     |     +-- (local-or-keystore)
       |     |        ...
       |     +--:(tls12-psk)
       |     |        {server-ident-tls12-psk,not tlss:server-ident-tls12-psk}?
       |     |  +-- tls12-psk
       |     |     +-- (local-or-keystore)
       |     |     |     ...
       |     |     +-- id_hint?                    string
```

```
|     +--:(tls13-epsk) {server-ident-tls13-epsk}?
|        +-- tls13-epsk
|           +-- (local-or-keystore)
|           |     ...
|           +-- external-identity          string
|           +-- hash
|           |       tlscmn:epsk-supported-hash
|           +-- context?                   string
|           +-- target-protocol?           uint16
|           +-- target-kdf?                uint16
|  +-- client-authentication! {client-auth-supported}?
|  |  +-- ca-certs! {client-auth-x509-cert}?
|  |  |  +-- (local-or-truststore)
|  |  |     +--:(local) {local-definitions-supported}?
|  |  |     |  +-- local-definition
|  |  |     |        ...
|  |  |     +--:(truststore)
|  |  |              {central-truststore-supported,certificates}?
|  |  |        +-- truststore-reference?   ts:certificate-bag-ref
|  |  +-- ee-certs! {client-auth-x509-cert}?
|  |  |  +-- (local-or-truststore)
|  |  |     +--:(local) {local-definitions-supported}?
|  |  |     |  +-- local-definition
|  |  |     |        ...
|  |  |     +--:(truststore)
|  |  |              {central-truststore-supported,certificates}?
|  |  |        +-- truststore-reference?   ts:certificate-bag-ref
|  |  +-- raw-public-keys! {client-auth-raw-public-key}?
|  |  |  +-- (local-or-truststore)
|  |  |     +--:(local) {local-definitions-supported}?
|  |  |     |  +-- local-definition
|  |  |     |        ...
|  |  |     +--:(truststore)
|  |  |              {central-truststore-supported,public-keys}?
|  |  |        +-- truststore-reference?   ts:public-key-bag-ref
|  |  +-- tls12-psks?        empty
|  |  |       {client-auth-tls12-psk,not tlss:client-auth-tls12-psk}?
|  |  +-- tls13-epsks?       empty {client-auth-tls13-epsk}?
|  +-- hello-params {tlscmn:hello-params}?
|  |  +-- tls-versions
|  |  |  +-- tls-version*   identityref
|  |  +-- cipher-suites
|  |     +-- cipher-suite*   identityref
|  +-- keepalives {tls-server-keepalives}?
|     +-- peer-allowed-to-send?   empty
|     +-- test-peer-aliveness!
|        +-- max-wait?       uint16
|        +-- max-attempts?   uint8
```

3.3. YANG Module

The "ietf-udp-server" YANG module uses the groupings defined in [I-D.ietf-netconf-tls-client-server] for configuring the DTLS 1.3 encryption.

```
<CODE BEGINS> file "ietf-udp-server@2023-10-16.yang"
module ietf-udp-server {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-udp-server";
  prefix udps;
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-tls-server {
    prefix tlss;
    reference
      "RFC FFFF: YANG Groupings for TLS Clients and TLS Servers";
  }

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

     Authors:  Alex Huang Feng
               <mailto:alex.huang-feng@insa-lyon.fr>
               Pierre Francois
               <mailto:pierre.francois@insa-lyon.fr>";

  description
    "Defines a generic grouping for UDP-based server applications.
    Supports groupings for UDP servers and UDP servers with DTLS encryption.

    Copyright (c) 2023 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

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

    This version of this YANG module is part of RFC-to-be; see the RFC
    itself for full legal notices.";
```

```
     revision 2023-10-16 {
       description
         "Initial revision";
       reference
         "RFC-to-be: YANG Grouping for UDP Clients and UDP Servers";
     }

    /*
     * FEATURES
     */
     feature dtls13 {
       description
         "This feature indicates that DTLS 1.3 encryption of UDP
          packets is supported.";
     }

     grouping udp-server-grouping {
       description
         "Provides a reusable grouping for configuring a UDP servers.";

       leaf local-address {
         type inet:ip-address-no-zone;
         mandatory true;
         description
           "IP address of the UDP server, which can be an
           IPv4 address or an IPV6 address.";
       }

       leaf local-port {
         type inet:port-number;
         mandatory true;
         description
           "Port number of the UDP server.";
       }
     }

     grouping udp-dtls-server-grouping {
       description
         "Provides a reusable grouping for configuring a UDP server with
         DTLS encryption.";

       uses udp-server-grouping;
       container dtls {
         if-feature dtls13;
         presence dtls;
         uses tlss:tls-server-grouping {
           // Using tls-server-grouping without TLS1.2 parameters
           // allowing only DTLS 1.3
```

```
            refine "server-identity/auth-type/tls12-psk" {
              // create the logical impossibility of enabling TLS1.2
              if-feature "not tlss:server-ident-tls12-psk";
            }
            refine "client-authentication/tls12-psks" {
              // create the logical impossibility of enabling TLS1.2
              if-feature "not tlss:client-auth-tls12-psk";
            }
          }
          description
            "Container for configuring DTLS 1.3 parameters.";
        }
      }
    }
    <CODE ENDS>
```

4. Security Considerations

TODO:

5. IANA Considerations

This document describes the URIs from IETF XML Registry and the registration of a two new YANG module names

5.1. URI

IANA is requested to assign two new URI from the IETF XML Registry [RFC3688]. The following two URIs are suggested:

URI: urn:ietf:params:xml:ns:yang:ietf-udp-client
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-udp-server
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

5.2. YANG module name

This document also requests two new YANG module names in the YANG Module Names registry [RFC8342] with the following suggestions:

name: ietf-udp-client
namespace: urn:ietf:params:xml:ns:yang:ietf-udp-client
prefix: udpc
reference: RFC-to-be

```
name: ietf-udp-server
namespace: urn:ietf:params:xml:ns:yang:ietf-udp-server
prefix: udps
reference: RFC-to-be
```

6. Acknowledgements

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

7. References

7.1. Normative References

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tls-client-server-33, 17 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-33>.

[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>.

[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>.

[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>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

7.2. Informative References

Authors' Addresses

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

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

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

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 9 April 2024

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

7 October 2023

Subscription to Distributed Notifications
draft-ietf-netconf-distributed-notif-08

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", "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 9 April 2024.

Table of Contents

1. Introduction

The mechanism to support a subscription of a continuous and customized stream of updates from a YANG datastore [RFC8342] 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 complements 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.

Node: is the Publisher that obtains and pushes the data to the Receiver.

Message Publisher: is the Publisher that pushes the message to the Receiver.

Message Publisher ID: A 32-bit identifier of the publishing process that is locally unique to the publisher node. With this identifier the publishing process from where the message was published from can be uniquely identified. Receivers SHOULD use the transport session and the Publisher ID field to separate different publisher streams originating from the same network node.

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 Message Publisher 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 1 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 wants 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.

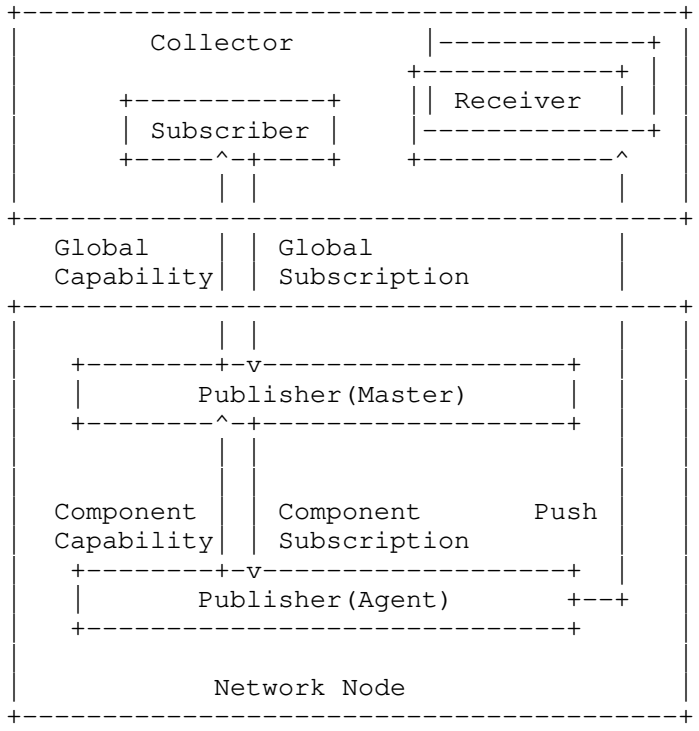


Figure 1: The Distributed Data Export Framework

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 Publisher ID to one of the publisher 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 Publisher 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 Publisher 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-publisher-id*   uint32
  augment /sn:subscription-started:
    +--ro message-publisher-id*   uint32
  augment /sn:subscription-modified:
    +--ro message-publisher-id*   uint32
  augment /sn:establish-subscription/sn:output:
    +--ro message-publisher-id*   uint32
```

10. YANG Module

```
<CODE BEGINS> file "ietf-distributed-notif@2023-09-17.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 2023-09-17 {
    description
      "Initial version";
    reference
      "RFC XXXX: Subscription to Distributed Notifications";
  }

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

    leaf-list message-publisher-id {
      type uint32;
      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
      Publisher ID to be exposed for a subscription.";

    uses message-publisher-ids;
  }

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

    uses message-publisher-ids;
  }

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

```
      uses message-publisher-ids;
    }

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

      uses message-publisher-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. Implementation Status

Note to the RFC-Editor: Please remove this section before publishing.

12.1. Open Source Publisher

INSA Lyon implemented this document for a YANG Push publisher on UDP-based Transport for Configured Subscriptions [I-D.ietf-netconf-udp-notif] in an example implementation.

The open source code can be obtained here: [INSA-Lyon-Publisher].

12.2. Open Source Receiver Library

INSA Lyon implemented this document for a YANG Push receiver on UDP-based Transport for Configured Subscriptions [I-D.ietf-netconf-udp-notif] as a library.

The open source code can be obtained here: [INSA-Lyon-Receiver].

12.3. Pmacct Data Collection

The open source YANG push receiver library has been integrated into the Pmacct open source Network Telemetry data collection.

12.4. Huawei VRP

Huawei implemented this document for a YANG Push publisher on UDP-based Transport for Configured Subscriptions [I-D.ietf-netconf-udp-notif] in their VRP platform.

13. 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 particulary for 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-publisher-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].

14. Contributors

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

15. Acknowledgements

We thank Kent Watsen, Mahesh Jethanandani, Martin Bjorklund, Tim Carey, Qin Wu, Robert Wilton, Benoit Claise and Alex Huang Feng for their constructive suggestions for improving this document.

16. References

16.1. 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>.

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

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

[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>.

16.2. 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-13, 4 November 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-13>.

[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-10, 7 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-udp-notif-10>.

[INSA-Lyon-Publisher]
"INSA Lyon, YANG Push publisher example implementation", <https://github.com/network-analytics/udp-notif-scapy>.

[INSA-Lyon-Receiver]
"INSA Lyon, YANG Push receiver library implementation", <https://github.com/network-analytics/udp-notif-c-collector>.

[Paolo-Lucente-Pmacct]
"Paolo Lucente, Pmacct open source Network Telemetry Data Collection", <https://github.com/pmacct/pmacct>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

Appendix A. Examples

This appendix is non-normative.

A.1. Dynamic Subscription

Figure 2 shows a typical dynamic subscription to the network node with distributed data export capability.

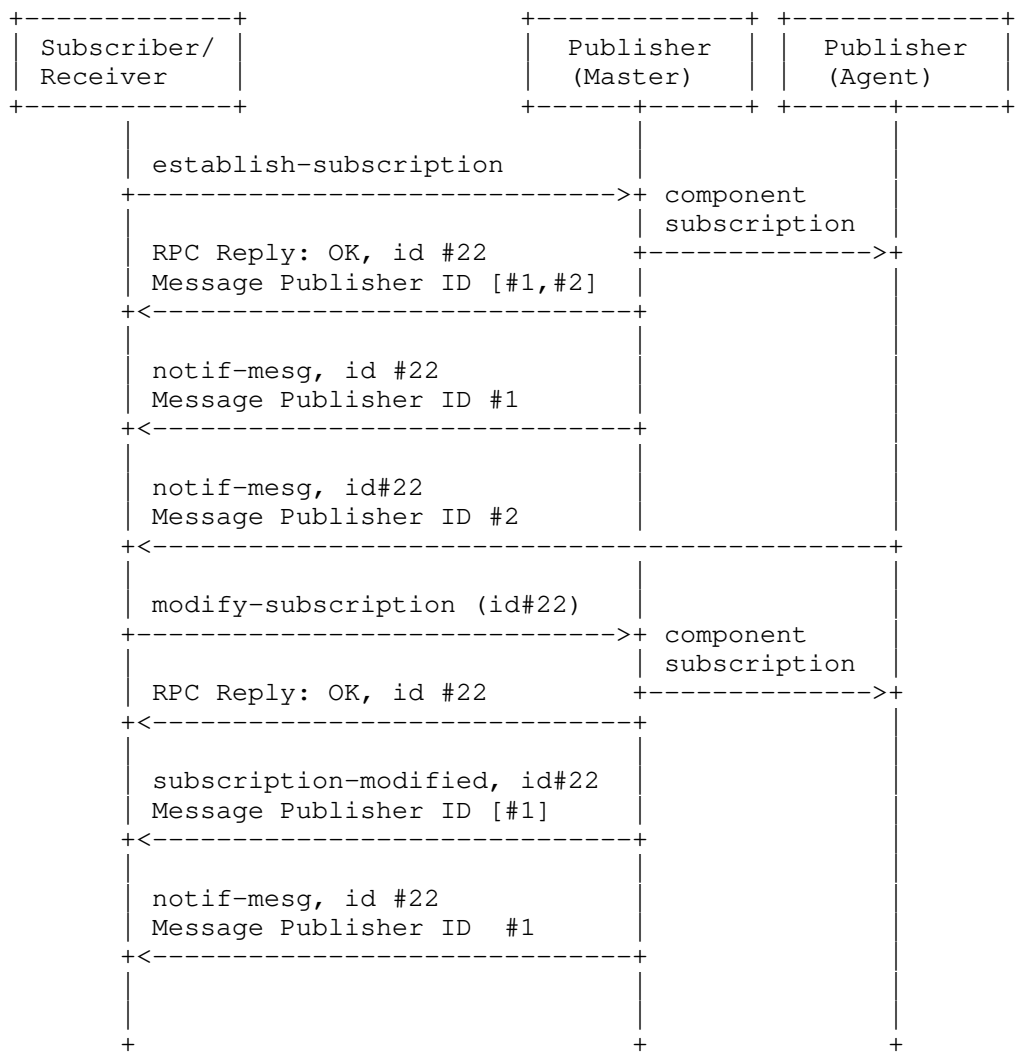


Figure 2: Call Flow for Dynamic Subscription

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: "establish-subscription" Request

As the network node is able to fully satisfy the request, the request is given a subscription ID of 22. The response as in Figure 4 indicates that the subscription is decomposed into two component subscriptions which will be published by two message Message Publisher 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-publisher-id
   xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
     1
  </message-publisher-id>
  <message-publisher-id
   xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
     2
  </message-publisher-id>
</rpc-reply>
```

Figure 4: "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 5 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: "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 6 for example, indicates the modified subscription is decomposed into one component subscription which will be published by message Message Publisher ID #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-publisher-id
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notificationss>
       1
    </message-publisher-id>
  </subscription-modified>
</notification>
```

Figure 6: "subscription-modified" Subscription State Notification

A.2. Configured Subscription

Figure 7 shows a typical configured subscription to the network node with distributed data export capability.

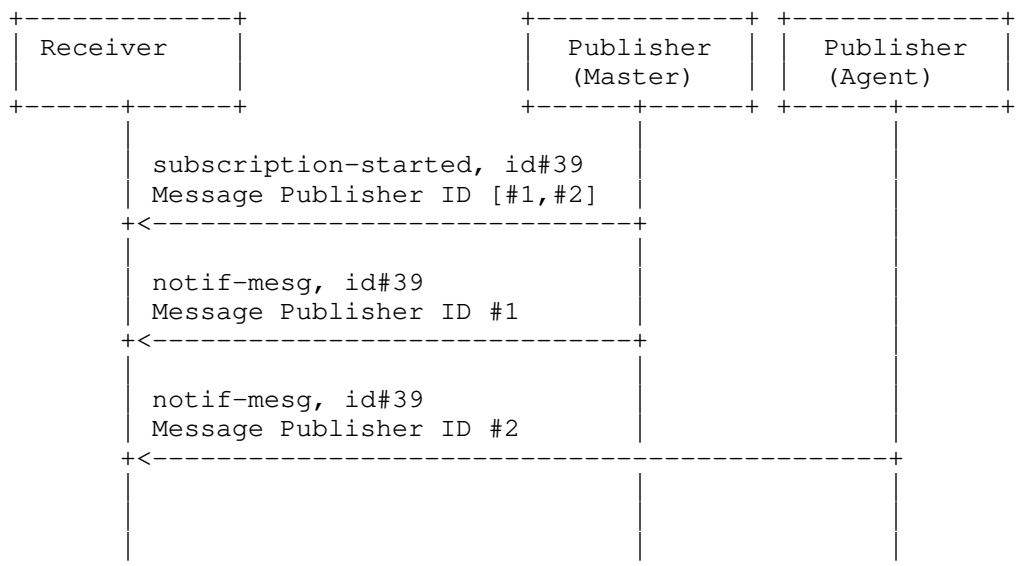


Figure 7: Call Flow for Configured Subscription

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 Message Publisher IDs: #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-publisher-id
     xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
        1
     </message-publisher-id>
     <message-publisher-id
     xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications>
        2
     </message-publisher-id>
   </subscription-started>
 </notification>
```

Figure 8: "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: 12 September 2023

K. Watsen
Watsen Networks
Q. Wu
Huawei Technologies
O. Hagsand
Netgate
H. Li
Hewlett Packard Enterprise
P. Andersson
Cisco Systems
11 March 2023

List Pagination for YANG-driven Protocols
draft-ietf-netconf-list-pagination-01

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), and either a combination of the "offset" parameter (see Section 3.1.4) or the "cursor" parameter (see Section 3.1.5), and lastly "the "limit" parameter (see Section 3.1.6).

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 "cursor" Query Parameter

Description
The "cursor" query parameter indicates where to start the working result-set (i.e., after the "direction" parameter has been applied), the elements before the cursor are skipped over when preparing the response. Furthermore the result-set is annotated with attributes for the next and previous cursors following a result-set constrained with the "limit" query parameter.

Default Value
If this query parameter is unspecified, then no entries in the result-set are skipped.

Allowed Values
The allowed values are base64 encoded positions interpreted by the server to index an element in the list. It is an error to supply an unkown cursor for the working result-set, and the "cursor-not-found" identity SHOULD be produced in the error output when this occurs.

Conformance
The "cursor" query parameter MUST be supported for all lists.

3.1.6. 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 3.1.6) or "sublist-limit" parameters (Appendix A.3.7).

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@2023-03-11.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 2023-03-11 {
    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.";
    }

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

    // 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 cursor-param-grouping {
        description
          "This grouping may be used by protocol-specific YANG modules
           to define a protocol-specific query parameter.";
        leaf cursor {
          type string;
          description
            "The 'cursor' parameter indicates where to start the working
             result-set (i.e. after the 'direction' parameter has been
             applied), the elements before the cursor are skipped over
             when preparing the response. Furthermare the result-set is
             annotated with attributes for the next and previous cursors
             following a result-set constrained with the 'limit' query
             parameter.";
        }
      }

      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 2023-03-11 {
  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 "cursor" Parameter

Noting that "cursor" must be an base64 encoded opaque value which addresses an element in a list.

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

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

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

A.3.3.1. cursor=&limit=2

```
REQUEST

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

RESPONSE

{
  "example-social:member": [
    {
      "member-id": "bob",
      "email-address": "bob@example.com",
      "password": "$0$1543",
      "avatar": "BASE64VALUE=",
```

```
        "tagline": "Here and now, like never before.",
        "posts": {
          "post": [
            {
              "timestamp": "2020-08-14T03:32:25Z",
              "body": "Just got in."
            },
            {
              "timestamp": "2020-08-14T03:33:55Z",
              "body": "What's new?"
            },
            {
              "timestamp": "2020-08-14T03:34:30Z",
              "body": "I'm bored..."
            }
          ]
        },
        "favorites": {
          "decimal64-numbers": ["3.14159", "2.71828"]
        },
        "stats": {
          "joined": "2020-08-14T03:30:00Z",
          "membership-level": "standard",
          "last-activity": "2020-08-14T03:34:30Z"
        }
      },
      {
        "member-id": "eric",
        "email-address": "eric@example.com",
        "password": "$0$1543",
        "avatar": "BASE64VALUE=",
        "tagline": "Go to bed with dreams; wake up with a purpose.",
        "following": ["alice"],
        "posts": {
          "post": [
            {
              "timestamp": "2020-09-17T18:02:04Z",
              "title": "Son, brother, husband, father",
              "body": "What's your story?"
            }
          ]
        },
        "favorites": {
          "bits": ["two", "one", "zero"]
        },
        "stats": {
          "joined": "2020-09-17T19:38:32Z",
          "membership-level": "pro",
```

```
          "last-activity": "2020-09-17T18:02:04Z"
        }
      }
    ],
    "@example-social:member": [
      {
        "ietf-list-pagination:remaining": 3,
        "ietf-list-pagination:previous": "",
        "ietf-list-pagination:next": "YWxpY2U=" // alice
      }
    ]
  }
```

A.3.3.2. cursor="YWxpY2U="&limit=2

REQUEST

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

RESPONSE

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

```
            {
              "timestamp": "2020-07-09T01:32:23Z",
              "title": "Sleepy...",
              "body": "Catch y'all tomorrow."
            }
          ]
        },
        "favorites": {
          "uint8-numbers": [17, 13, 11, 7, 5, 3],
          "int8-numbers": [-5, -3, -1, 1, 3, 5]
        },
        "stats": {
          "joined": "2020-07-08T12:38:32Z",
          "membership-level": "admin",
          "last-activity": "2021-04-01T02:51:11Z"
        }
      },
      {
        "member-id": "lin",
        "email-address": "lin@example.com",
        "password": "$0$1543",
        "privacy-settings": {
          "hide-network": true,
          "post-visibility": "followers-only"
        },
        "following": ["joe", "eric", "alice"],
        "stats": {
          "joined": "2020-07-09T12:38:32Z",
          "membership-level": "standard",
          "last-activity": "2021-04-01T02:51:11Z"
        }
      }
    ],
    "@example-social:member": [
      {
        "ietf-list-pagination:remaining": 1,
        "ietf-list-pagination:previous": "ZXJpYw==", // eric
        "ietf-list-pagination:next": "am9l" // joe
      }
    ]
  }
```

A.3.3.3. cursor="am9l"&limit=2

REQUEST

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

   RESPONSE

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

A.3.4. The "direction" Parameter

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

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

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

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

A.3.4.1. direction=forwards

REQUEST

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

RESPONSE

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

A.3.4.2. direction=backwards

REQUEST

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

RESPONSE

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

A.3.5. The "sort-by" Parameter

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

A.3.5.1. the target node's type

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

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

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

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

```
REQUEST

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

RESPONSE
```

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

A.3.5.1.2. type is a "list" and sort-by node is a direct 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.3.5.1.3. 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.3.5.2. 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.3.6. 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.3.6.1. match of leaf-list’s values

FIXME

A.3.6.2. 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.3.6.3. match on decendent timestamp starting with a substring

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

REQUEST

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

RESPONSE

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

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

A.3.7. The "sublist-limit" Parameter

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

A.3.7.1. 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.3.7.2. 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.3.8. Combinations of Parameters

A.3.8.1. 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 Björklund, 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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 22 April 2024

JG. Cumming
Nokia
R. Wills
Cisco Systems

20 October 2023

NETCONF Private Candidates
draft-ietf-netconf-privcand-01

Abstract

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 22 April 2024.

Copyright Notice

Table of Contents

1. Introduction

NETCONF [RFC6241] and RESTCONF [RFC8040] both provide a mechanism for one or more clients to make configuration changes to a device running as a NETCONF/RESTCONF server. Each client has the ability to make one or more configuration 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 behavior may be undesirable as one client may unwittingly commit the configuration changes made by another client.

NETCONF provides a way to mitigate this behavior 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 behavior 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. It also describes how the RESTCONF protocol can be used on a system that implements private candidates.

1.1. Requirements Language

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

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.

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

When a private candidate is used by RESTCONF, it exists only for the duration of the RESTCONF request.

The private candidate configuration contains a full copy of the running configuration when it is created (in the same way as a branch does in a source control management system and in the same way as the candidate configuration datastore as defined in [RFC6241]). Any changes made to it, for example, through the use of operations such as <edit-config> and <edit-data>, 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). A <discard-changes> operation will revert the private candidate to the branch's initial state or it's state at the last <commit> (whichever is most recent).

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

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 behavior 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.

Finally, this locking mechanism isn’t available to RESTCONF clients.

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 and RESTCONF clients as each configuration change is applied immediately and the client has no knowledge of the current configuration at the point in time that they commenced the editing activity nor at the point they commit the activity.

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. Private candidates solution

The use of private candidates resolves the issues detailed earlier in this document.

NETCONF sessions and RESTCONF requests 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 SHOULD:

* Be accessible by one client only

* Be visible by one client only

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

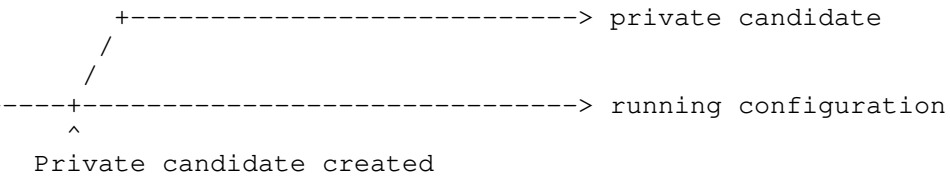
4.1. What is a private candidate

A private candidate is defined earlier in the definitions and terminology section of this document.

4.2. When is a private candidate created

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

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



4.3. When is a private candidate destroyed

A private candidate is valid for the duration of the NETCONF session. Issuing a <commit> operation will not close the private candidate but will issue an implicit <update> operation resyncing changes from the running configuration. More details on this later in this document.

A NETCONF session that is operating using a private candidate will discard all uncommitted changes in that session's private candidate and destroy the private candidate if the session is closed through a deliberate user action or disconnected for any other reason (such as a loss of network connectivity).

4.4. How to signal the use of private candidates

4.4.1. Server

The server MUST signal its support for private candidates. The server does this by advertising a new :private-candidate capability:

urn:ietf:params:netconf:capability:private-candidate:1.0

A server may also advertise the :candidate capability as defined in [RFC6241] if the shared candidate is also supported.

A non-NMDA capable NETCONF server that advertises the :private-candidate capability MUST also advertise the :candidate capability.

If the server has not signalled the :private-candidate capability, or otherwise does not support private candidates, the server MUST:

* Terminate the session when it receives the :private-candidate capability from a client in a <hello> message,

* Return an <rpc-error> if a client attempts to interact with the NMDA private-candidate configuration datastore.

4.4.2. NETCONF client

In order to utilise a private candidate configuration within a NETCONF session, the client must inform the server that it wishes to do this.

Two approaches are available for a NETCONF client to signal that it wants to use a private candidate:

4.4.2.1. Client capability declaration

When a NETCONF client connects with a server it sends a list of client capabilities including one of the :base NETCONF version capabilties.

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

urn:ietf:params:netconf:capability:private-candidate:1.0

In order for the use of private candidates to be established using this approach both the NETCONF server and the NETCONF client MUST advertise this capability.

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

All RPC requests that target the candidate configuration datastore will operate in exactly the same way as they would do when using the shared candidate configuration datastore, however, when the server receives a request to act upon the candidate configuration datastore it instead uses the session's private candidate configuration datastore.

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 any RPCs (including <lock>)

4.4.2.2. Private candidate datastore

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

All NMDA operations that support candidate NMDA datastore 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 server to support private candidates is optional and SHOULD be signalled in NMDA supporting servers as a datastore in addition to the server capabilities described earlier in this document.

To use this method the client is not required to send the :private-candidate capability.

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 MUST fail.

4.4.3. RESTCONF client

RESTCONF doesn't provide a mechanism for the client to advertise a capability. Therefore when a RESTCONF server advertises the :private-candidate capability, the decision of whether to use a private candidate depends on whether a datastore is explicitly referenced in the request using the RESTCONF extensions for NMDA [RFC8527].

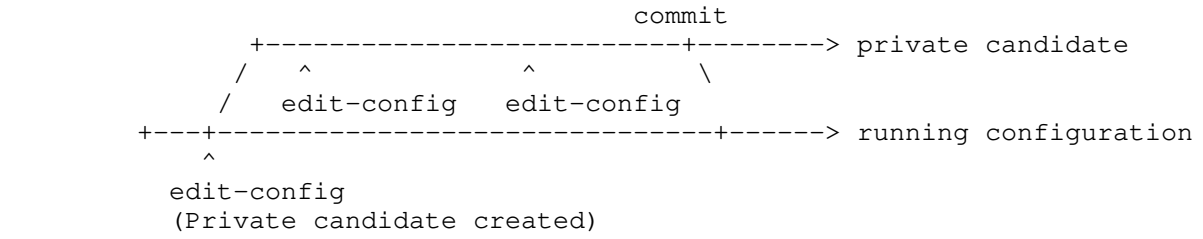
When the server advertises the :private-candidate capability and the client does not explicitly reference a datastore in their request, all edits are made to a new private candidate, and the private candidate is committed. This is analagous to the behavior of RESTCONF when the :candidate capability is specified by the server.

When the private-candidate datastore is explicitly referenced, edits are made to a new private candidate and the private candidate is committed.

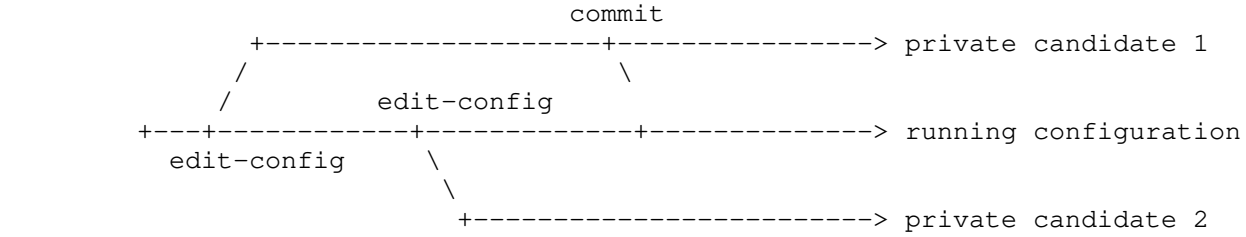
4.5. Interaction between running and private-candidate(s)

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

In the simplest example, a session may create a private candidate configuration, perform multiple NETCONF operations (such as <edit-config>) on it and then perform a <commit> operation to merge the private candidate configuration into the running configuration in line with semantics in [RFC6241].



More complex scenarios need to be considered, when multiple private candidate sessions are working on their own configuration (branches) and they make commits into the running configuration.



In this situation, if, how and when private candidate 2 is updated with the information that the running configuration has changed must be considered.

As described earlier, the client MUST be aware of changes to it's private candidate configuration so it can be assured that it is only committing its own modifications. It should also be aware of any changes to the current running configuration.

It is possible, during an update, for conflicts to occur and the detection and resolution of these is discussed later in this document.

Two modes of operation are provided. Both modes may be supported by a system, however, only one mode MUST be supported per session.

The server MUST advertise which mode is being used by the session by providing the mode parameter to the :private-candidate capability.

4.5.1. Static branch mode: Independent private candidate branch

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 (which MUST automatically issue an <update> operation immediately prior to commiting the configuration)

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 creation point of the private candidate's branch, against the last update point of the private candidate's branch or against the running configuration.

Conflict detection and resolution is discussed later in this document.

The server signals this mode by setting the mode parameter to the :private-candidate capability to static-branch as follows:

urn:ietf:params:netconf:capability:private-candidate:1.0
?mode=static-branch

This is the default mode. If no mode is specified in the :private-candidate capability this mode is used.

4.5.2. Continuous rebase mode: Continually updating private candidate

The private candidate is treated as a separate branch, however, when any change is made to the running configuration the update operation will automatically be run on all open private candidate branches.

This is equivalent to all currently open private candidate branches being rebased onto the running configuration every time a change is made to it by any session.

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 creation point of the private candidate's branch, against the last update point of the private candidate's branch or against the running configuration. The output of the <compare> operation may be identical when comparing the current position of the private candidate with the last updated point or the running configuration depending on the resolution mode discussed below.

* The output of the <compare> operation may not match the set of changes made to the session's private candidate by the sessions owner but may also include 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. For this reason restrictions are placed on what resolution modes are available for these automated updates.

The server signals this mode by setting the mode parameter to the :private-candidate capability to continuous-rebase as follows:

urn:ietf:params:netconf:capability:private-candidate:1.0
?mode=continuous-rebase

Conflict detection and resolution is discussed later in this document.

4.6. Detecting and resolving conflicts

4.6.1. What is a conflict?

A conflict is when the intent of the NETCONF client may have been different had it had a different starting point. In configuration terms, a conflict occurs when the same set of nodes in a configuration being altered by one user are changed between the start of the configuration preparation (the first <edit-config>/<edit-data> operation) and the conclusion of this configuration session (terminated by a <commit> operation).

The situation where conflicts have the potential of occurring are when multiple configuration sessions are in progress and one session commits changes into the running configuration after the private candidate (branch) was created.

When this happens a conflict occurs if the nodes modified in the running configuration are the same nodes that are modified in the private candidate configuration.

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.

4.6.2. Detecting and reporting conflicts

A conflict can occur when an <update> operation is triggered. This can occur in a number of ways:

* Manually triggered by the <update> NETCONF operation

* Automatically triggered by the NETCONF server running an <update> operation upon a <commit> being issued by the client in the private candidate session.

* Automatically triggered by the NETCONF server running an <update> operation upon a <commit> being issued by any other configuration session (or user). This occurs in continual rebase mode only.

When a conflict occurs the client MUST be given the opportunity to re-evaluate its intent based on the new information. The resolution of the conflict may be manual or automatic depending on the server and client decision (discussed later in this document).

When a conflict occurs, a <commit> or <update> operation MUST fail. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

In continuous rebase mode, it is theoretically possible for the automated <update> operation to fail. To mitigate against this (as the client cannot be provided this information), restrictions are placed on the resolution methods allowed for the automated update operation.

The location of the conflict(s) should be reported as a list of xpaths and values.

4.6.3. Conflict resolution

Conflict resolution defines which configuration elements are retained when a conflict is resolved; those from the running configuration or those from the private candidate configuration.

When a conflict is detected in any client triggered activity, the client MUST be informed. The client then has a number of options available to resolve the conflict.

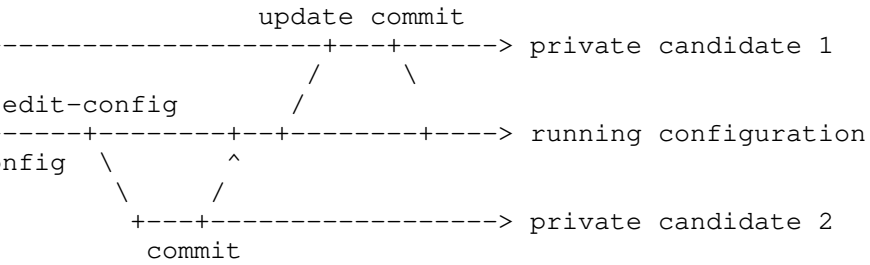
It is worth noting that in the case of continuous rebase mode automated <update> operations may be performed against multiple private candidate configurations at once.

The resolution method SHOULD be provided as an input to the <update> operation described later in this document. This input may be through a default selection, a specific input or a configuration element.

The following configuration data is used below to describe the behavior of each resolution method:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to London<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link to Tokyo<description>
    </interface>
  </interfaces>
</configure>
```

The following workflow diagram is used and the outcome is the same regardless of whether static branch mode or continuous rebase mode is being used. For the purpose of the examples below assume the update operation is manually provided by a client in static branch mode.



There are three defined resolution methods:

4.6.3.1. Ignore

When using the ignore resolution method items in the running configuration that are not in conflict with the private candidate configuration are merged from the running configuration into the private candidate configuration. Nodes that are in conflict are ignored and not merged. The outcome of this is that the private candidate configuration reflects changes in the running that were not being worked on and those that are being worked on in the private candidate remain in the private candidate. Issuing a <commit> operation at this point will overwrite the running configuration with the conflicted items from the private candidate configuration.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and commits the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>ignore</resolution-mode>
  </update>
</rpc>
```

The un-conflicting changes are merged and the conflicting ones are ignored (and not merged from the running into private candidate 1).

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to San Francisco<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link moved to Paris<description>
    </interface>
  </interfaces>
</configure>
```

4.6.3.2. Overwrite

When using the overwrite resolution method items in the running configuration that are not in conflict with the private candidate configuration are merged from the running configuration into the private candidate configuration. Nodes that are in conflict are pushed from the running configuration into the private candidate configuration, overwriting any previous changes in the private candidate configuration. The outcome of this is that the private candidate configuration reflects the changes in the running configuration that were not being worked on as well as changing those being worked on in the private candidate to new values.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and commits the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>overwrite</resolution-mode>
  </update>
</rpc>
```

The un-conflicting changes are merged and the conflicting ones are pushed into the private candidate 1 overwriting the existing changes.

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_two</name>
      <description>Link moved to Paris<description>
    </interface>
  </interfaces>
</configure>
```

4.6.3.3. Revert-on-conflict

When using the revert-on-conflict resolution method an update will fail to complete when any conflicting node is found. The session issuing the update will be informed of the failure.

No changes, whether conflicting or un-conflicting are merged into the private candidate configuration.

The owner of the private candidate session must then take deliberate and specific action to adjust the private candidate configuration to rectify the conflict. This may be by issuing further <edit-config> or <edit-data> operations, by issuing a <discard-changes> operation or by issuing an <update> operation with a different resolution method.

This resolution method is the default resolution method as it provides for the highest level of visibility and control to ensure operational stability.

This resolution method may not be selected by a system operating in continuous rebase mode when performing automatic <update> operations. Clients operating in continuous rebase mode may use this resolution mode in their <update> operation.

Example:

Session 1 edits the configuration by submitting the following

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name>intf_one</name>
            <description>Link to San Francisco<description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and commits the configuration to the running configuration datastore.

```
<rpc message-id="config"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target><candidate/><target>
    <config>
      <configure>
        <interfaces>
          <interface>
            <name operation="delete">intf_one</name>
          </interface>
          <interface>
            <name>intf_two</name>
            <description>Link moved to Paris</description>
          </interface>
        </interfaces>
      </configure>
    </config>
  </edit-config>
</rpc>
```

Session 1 then sends an <update> NETCONF operation.

```
<rpc message-id="update"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <update>
    <resolution-mode>revert-on-conflict</resolution-mode>
  </update>
</rpc>
```

A conflict is detected, the update fails with an <rpc-error> and no merges/overwrite operations happen.

The resulting data in private candidate 1 is:

```
<configure>
  <interfaces>
    <interface>
      <name>intf_one</name>
      <description>Link to San Francisco<description>
    </interface>
    <interface>
      <name>intf_two</name>
      <description>Link to Tokyo<description>
    </interface>
  </interfaces>
</configure>
```

4.6.4. Default resolution mode and advertisement of this mode

The default resolution mode is revert-on-conflict, however, a system MAY choose to select a different default resolution mode.

The default resolution mode MAY be advertised in the :private-candidate capability by adding the resolution-mode parameter. If the system default is revert-on-conflict then this is optional.

If a server does not support revert-on-conflict it MUST report the default resolution mode.

If the system default is chosen to be anything other than revert-on-conflict then this MUST be signalled using the resolution-mode parameter, for example:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?mode=static-branch&default-resolution-mode=overwrite
```

4.6.5. Supported resolution modes

A server SHOULD support all three resolution modes, however, if the server does not support all three modes, the server MUST report the supported modes in the :private-candidate capability using the supported-resolution-modes, for example:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?mode=static-branch
              &supported-resolution-modes=revert-on-conflict,ignore
```

4.7. NETCONF operations

4.7.1. New NETCONF operations

4.7.1.1. <update>

The <update> operation is provided to allow NETCONF clients (or servers) to trigger a rebase of the private candidate configuration against the running configuration.

The <update> operation may be triggered manually by the client or automatically by the server.

The <update> operation MUST be triggered on all private candidates by a <commit> operation being executed in any candidate configuration on the device if the device is operating in continuous rebase mode.

The <update> operation MUST be implicitly triggered by a specific NETCONF session issuing a <commit> operation when using private candidates.

The actual order of operations in the server MUST be to issue the implicit <update> operation first and then the <commit> operation.

A <commit> operation that fails the implicit <update> operation SHOULD fail. The client is then required to make a specific decision to rectify the issue prior to committing. This may be to edit the private candidate configuration or to issue a manual <update> operation with a specific resolution mode selected.

4.7.1.1.1. <resolution-mode> parameter

The <update> operation takes the optional <resolution-mode> parameter

The resolution modes are described earlier in this document and the accepted inputs are:

* revert-on-conflict (default)

* ignore

* overwrite

4.7.2. Updated NETCONF operations

Specific NETCONF operations altered by this document are listed in this section. Any notable behavior with existing unaltered NETCONF operations is noted in the appendix.

4.7.2.1. <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.

4.7.2.2. <lock> and <unlock>

Performing a <lock> on the private-candidate datastore is a valid operation, although it is understood that the practical effect of this is a 'no op' as only one session may edit the locked private candidate.

If the client's intention is that no other session may commit changes to the system then the client should issue a <lock> operation on the running candidate.

Other NETCONF sessions are still able to create a new private-candidate configurations, make edits to them and perform operations on them, such as <update> or <discard-changes>.

Performing an <unlock> on the private-candidate datastore is a valid operation

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

4.7.2.3. <compare>

Performing a <compare> [RFC9144] operation 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 is extended by this document to allow the ability to compare the private-candidate datastore (at its current point in time) with the same private-candidate datastore at an earlier point in time or with another datastore.

4.7.2.3.1. <reference-point> parameter

This document adds the optional <reference-point> node to the input of the <compare> operation that accepts the following values:

* last-update

* >creation-point

Servers MAY support this functionality but are not required to by this document.

The last-update selection of <reference-point> will provide an output comparing the current private-candidate configuration datastore with the same private-candidate datastore at the time it was last updated using the <update> NETCONF operation described in this document (whether automatically or manually triggered).

The creation-point selection of <reference-point> will provide an output comparing the current private-candidate configuration datastore with the same private-candidate datastore at the time this private-candidate was initially created.

4.7.2.4. <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.

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

4.7.2.5. <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.

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

4.7.2.6. <copy-config>

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

4.7.2.7. <delete-config>

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

4.7.2.8. <commit>

The <commit> operation MUST trigger an implicit <update> operation.

Nothing in this document alters the standard behavior of the <persist> or <persist-id> options and these SHOULD work when using the private-candidate configuration datastore.

5. IANA Considerations

This document requests the registration the the following NETCONF capabilities:

* urn:ietf:params:netconf:capability:private-candidate:1.0 (Version 1.0)

6. Security Considerations

This document should not affect the security of the Internet.

7. References

7.1. 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>.

[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>.

[RFC8527] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", RFC 8527, DOI 10.17487/RFC8527, March 2019, <https://www.rfc-editor.org/info/rfc8527>.

7.2. Informative References

Appendix A. Behavior with unaltered NETCONF operations

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.

Contributors

The authors would like to thank Jan Lindblad, Lori-Ann McGrath, 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: 12 April 2024

J. Lindblad
Cisco Systems

10 October 2023

Transaction ID Mechanism for NETCONF
draft-ietf-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.

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

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

Status of This Memo

This Internet-Draft will expire on 12 April 2024.

Table of Contents

1. Introduction

When a NETCONF client wishes to initiate a new configuration transaction with a NETCONF server, a frequently occurring use case is for the client to find out if the configuration has changed since the client last communicated with the server. 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 transaction tag or timestamp for an entire configuration datastore or YANG subtree, and offer clients a

way to read and compare this tag or timestamp. If the tag or timestamp is unchanged, clients can avoid performing expensive operations. Such tags and timestamps are referred to as a transaction id (txid) in this document.

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, [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 configuration being untouched by others. This mechanism leverages [RFC7232] "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests".

This document defines similar functionality for NETCONF, [RFC6241], for config true data. It also ties this in with YANG-Push, [RFC8641], and "Comparison of Network Management Datastore Architecture (NMDA) Datastores", [RFC9144]. Config false data (operational data, state, statistics) is left out of scope from this document.

This document does not change the RESTCONF protocol in any way, and is carefully written to allow implementations to share much of the code between NETCONF and RESTCONF. Note that the NETCONF txid mechanism described in this document uses XML attributes, but the RESTCONF mechanism relies on HTTP Headers instead, and use none of the XML attributes described in this document, nor JSON Metadata (see [RFC7952]).

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], [RFC7950], [RFC7952], [RFC8040], [RFC8641], and [RFC9144].

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.

Transaction-id Mechanism A protocol implementation that fulfills the principles described in the first part, NETCONF Txid Extension (Section 3), of this document.

Txid Abbreviation of Transaction-id

C-txid Client side transaction-id, i.e. a txid value maintained or provided by a NETCONF client application.

S-txid Server side transaction-id, i.e. a txid value maintained or sent by a NETCONF server.

Txid History Temporally ordered list of txid values used by the server. Allows the server to determine if a given txid occurred more recently than another txid.

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. A similar extension is also defined for servers implememnting "Comparison of NMDA Datastores".

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. This document is therefore divided into a two parts; the first part discusses the txid mechanism in an abstract, protocol-neutral way. The second part, Txid Mechanisms (Section 4), then adds the protocol layer, and provides concrete encoding examples.

3.1. Use Cases

The common use cases for txid 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.

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 top level server side txid meta data value for each configuration datastore supported by the server. Server side txid is often abbreviated s-txid. 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".

Server implementors MAY use the YANG extension statement ietf-netconf-txid:versioned-node to inform potential clients about which YANG nodes the server maintains a txid value for. Another way to discover (a partial) set of Versioned Nodes is for a client to request the current configuration with txids. The returned configuration will then have the Versioned Nodes decorated with their txid values.

Regardless of whether the server declares the Versioned Nodes or not, the set of Versioned Nodes in the server's YANG tree MUST remain constant, except at system redefining events, such as software upgrades or entitlement installations or removals.

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 Nodes, 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, and their ancestors, 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 [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, or any kind of metadata that the server may maintain for YANG data tree nodes.

3.3. Initial Configuration Retrieval

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

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.

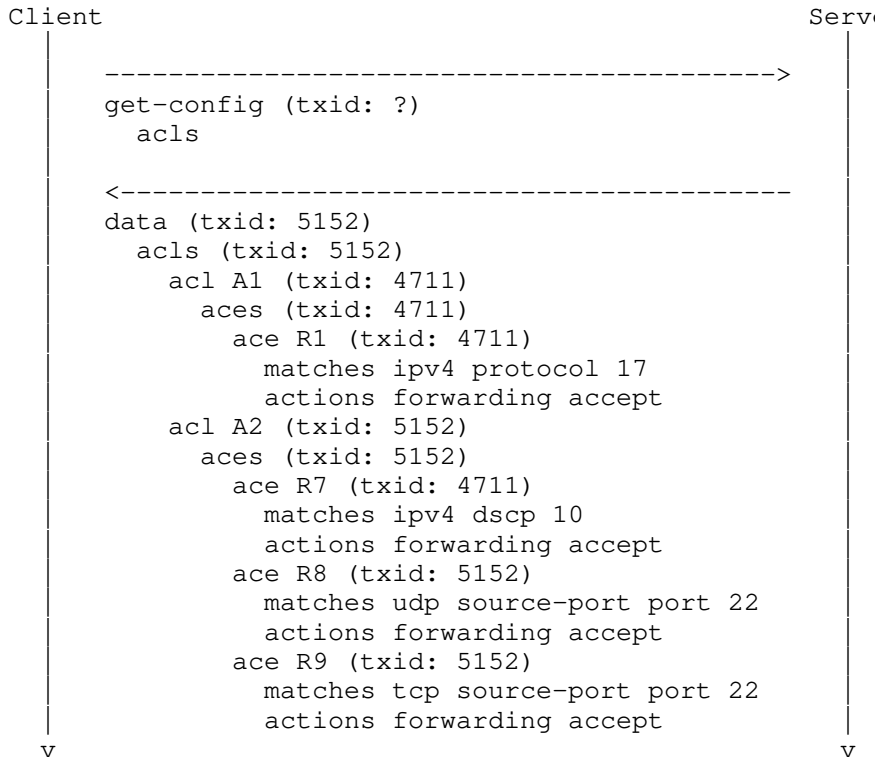


Figure 1: Initial Configuration Retrieval. The client annotated the get-config request itself with the txid request value, which makes the server return all txid values in the entire datastore, that also fall within the requested subtree filter. The most recent change seems to have been an update to ace R8 and R9.

In the call flow examples in this document we are using a 4-digit, monotonously increasing integer as txid. This is convenient and enhances readability of the examples, but does not necessarily reflect a typical implementation.

In principle, txid values are opaque strings that uniquely identify a particular configuration state. Servers are expected to know which txid values it has used in the recent past, and in which order they were assigned to configuration change transactions. This information is known as the server's Txid History.

How many historical txid values to track is up to each server implementor to decide, and a server MAY decide not to store any historical txid values at all. The more txid values in the server's Txid History, the more efficient the client synchronization may be, as described in the coming sections.

Some server implementors may decide to use a monotonically increasing integer as the txid value, or a timestamp. Doing so obviously makes it very easy for the server to determine the sequence of historical transaction ids.

Some server implementors may decide to use a completely different txid value sequence, to the point that the sequence may appear completely random to outside observers. Clients MUST NOT generally assume that servers use a txid value scheme that reveals information about the temporal sequence of txid values.

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. Txid values sent by a client are often abbreviated c-txid.

When a client sends in a c-txid value of a node that matches the server's s-txid value for that Versioned Node, or matches a more recent s-txid value in the server's Txid History, the server prunes (does not return) that subtree from the response. Since the client already knows the txid for this part of the data tree, or a txid that occurred more recently, it is obviosuly already up to date with that part of the configuration. Sending it again would be a waste of time and energy.

The table below describes in detail how the client side (c-txid) and server side txid (s-txid) values are determined and compared when the server processes each data tree reply node from a get-config or get-data request.

Servers MUST process each of the config true nodes as follows:

Case	Condition	Behavior
1. NO CLIENT TXID	In its request, the client did not specify a c-txid value for the current node, nor any ancestor of this node.	In this case, the server MUST return the current node according to the normal NETCONF specifications. The

		rules below do not apply to the current node. Any child nodes MUST also be evaluated with respect to these rules.
2. CLIENT ANCESTOR TXID	The client did not specify a c-txid value for the current node, but did specify a c-txid value for one or more ancestors of this node.	In this case, the current node MUST inherit the c-txid value of the closest ancestor node in the client's request that has a c-txid value. Processing of the current node continues according to the rules below.
3. SERVER ANCESTOR TXID	The node is not a Versioned Node, i.e. the server does not maintain a s-txid value for this node.	In this case, the current node MUST inherit the server's s-txid value of the closest ancestor that is a Versioned Node (has a server side s-txid value). The datastore root is always a Versioned Node. Processing of the current node continues according to the rules below.
4. CLIENT TXID UP TO DATE	The client specified c-txid for the current node value is "up to date", i.e. it matches the server's s-txid value, or matches a s-txid value from the server's Txid History that is more recent than the server's s-txid value for this node.	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.
5. CLIENT TXID OUT OF DATE	The specified c-txid is "outdated" or "unknown" to the server, i.e. it does not match the server's s-txid value for this node, nor does the client c-txid value match	In this case the server MUST return the current node according to the normal NETCONF specifications. If the current node is a Versioned Node, it MUST

	any s-txid value in the server's Txid History that is more recent than the server's s-txid value for this node.	be decorated with the s-txid value. Any child nodes MUST also be evaluated with respect to these rules.	

Table 1: The Txid rules for response pruning.

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.

Here follows a couple of examples of how the rules above are applied. See the example above (Figure 1) for the most recent server configuration state that the client is aware of, before this happens:

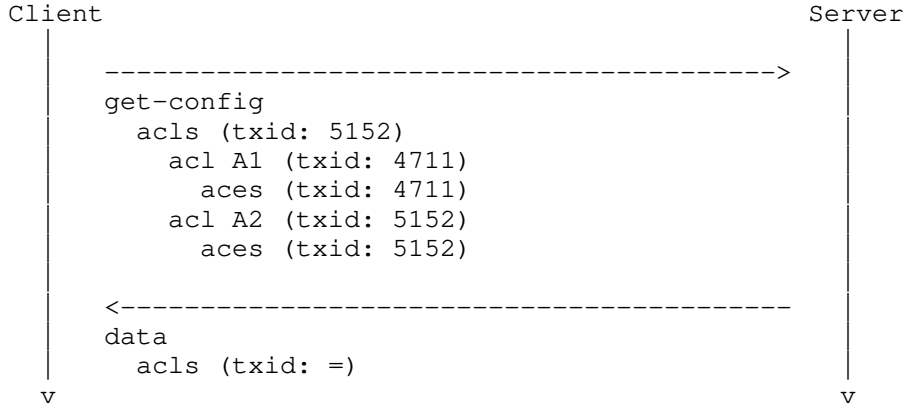


Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where the c-txid matches expectations. In this case, the server had no changes, and pruned the response at the earliest point offered by the client.

In this case, the server's txid-based pruning saved a substantial amount of information that is already known by the client to be sent to and processed by the client.

In the following example someone has made a change to the configuration on the server. This server has chosen to implement a Txid History with up to 5 entries. The 5 most recently used s-txid values on this example server are currently: 4711, 5152, 5550, 6614, 7770 (most recent). Then a client sends this request:

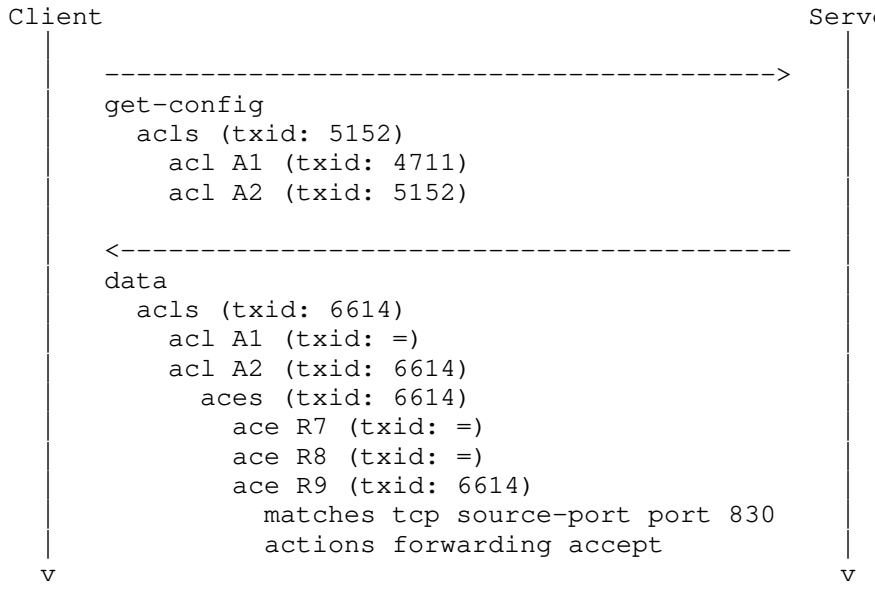


Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides updates only where changes have happened.

In the example above, the server returns the acls container because the client supplied c-txid value (5152) differs from the s-txid value held by the server (6614), and 5152 is less recent in the server's Txid History than 6614. The client is apparently unaware of the latest config developments in this part of the server config tree.

The server prunes list entry acl A1 is because it has the same s-txid value as the c-txid supplied by the client (4711). The server returns the list entry acl A2 because 5152 (specified by the client) is less recent than 6614 (held by the server).

The container aces under acl A2 is returned because 5152 is less recent than 6614. The server prunes ace R7 because the c-txid for this node is 5152 (from acl A2), and 5152 is more recent than the closest ancestor Versioned Node (with txid 4711).

The server also prunes acl R8 because the server and client txids exactly match (5152). Finally, acl R9 is returned because of its less recent c-txid value given by the client (5152, on the closest ancestor acl A2) than the s-txid held on the server (6614).

In the next example, the client specifies the c-txid for a node that the server does not maintain a s-txid for, i.e. it's not a Versioned Node.

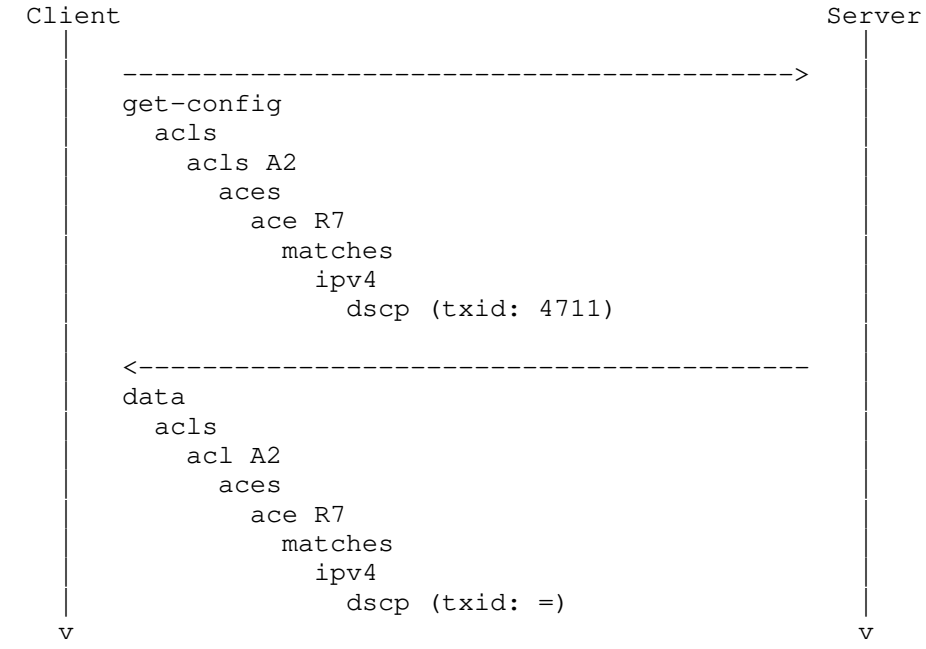


Figure 4: Versioned Nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the etag value is '=', and the leaf value is pruned.

Here, the server looks up the closest ancestor node that is a Versioned Node. This particular server has chosen to keep a s-txid for the list entry ace R7, but not for any of its children. Thus the server finds the server side s-txid value to be 4711 (from ace R7), which matches the client's c-txid value of 4711.

Servers MUST NOT ever use the special txid values, txid-match, txid-request, txid-unknown (e.g. "=", "?", "!") as actual txid values.

3.5. Candidate Datastore Configuration Retrieval

When a client retrieves the configuration from the (or a) candidate datastore, some of the configuration nodes may hold the same data as the corresponding node in the running datastore. In such cases, the server MUST return the same s-txid value for nodes in the candidate datastore as in the running datastore.

If a node in the candidate datastore holds different data than in the running datastore, the server has a choice of what to return.

* The server MAY return a txid-unknown value (e.g. "!"). This may be convenient in servers that do not know a priori what txids will be used in a future, possible commit of the canidate.

* If the txid-unknown value is not returned, the server MUST return the s-txid value the node will have after commit, assuming the client makes no further changes of the candidate datastore. If a client makes further changes in the candidate datastore, the s-txid value MAY change.

See the example in Candidate Datastore Transactions (Section 3.7).

3.6. 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 c-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 configuration 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 s-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 s-txid with the ok message.

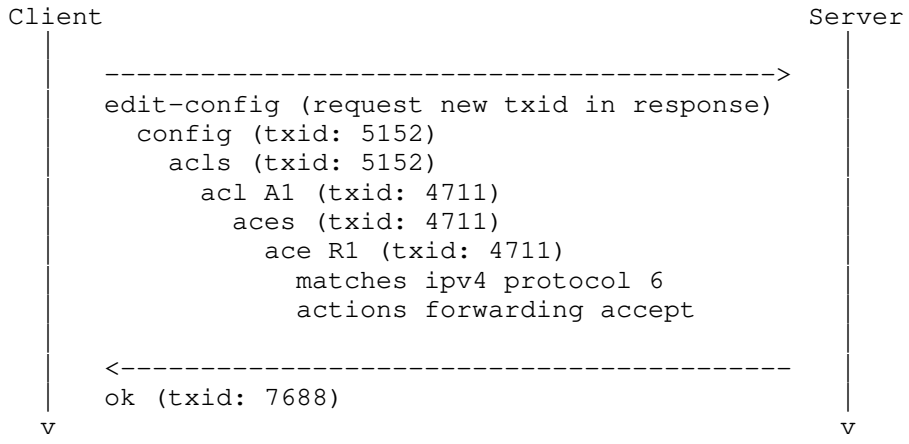


Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

After the above edit-config, the client might issues a get-config to observe the change. It would look like this:

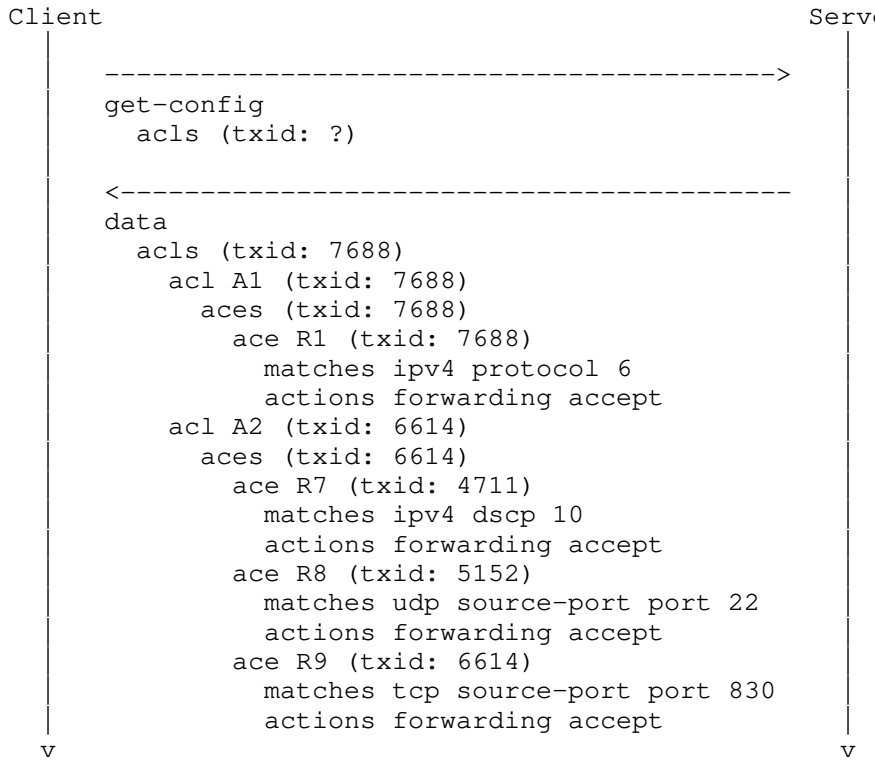


Figure 6: The txids are updated on all Versioned Nodes that were modified themselves or have a child node that was modified.

When a client sends in a c-txid value of a node, the server MUST consider it a match if the server's s-txid value is identical to the client, or if the server's value is found earlier in the server's Txid History than the value supplied by the client.

3.6.1. Error response on Out of band change

If the server rejects the transaction because one or more of the configuration s-txid value(s) differs from the client's expectation, the server MUST return at least one rpc-error with the following values:

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

Additionally, the error-info tag MUST contain an sx:structure containing relevant details about one of the mismatching txids. A server MAY send multiple rpc-errors when multiple txid mismatches are detected.

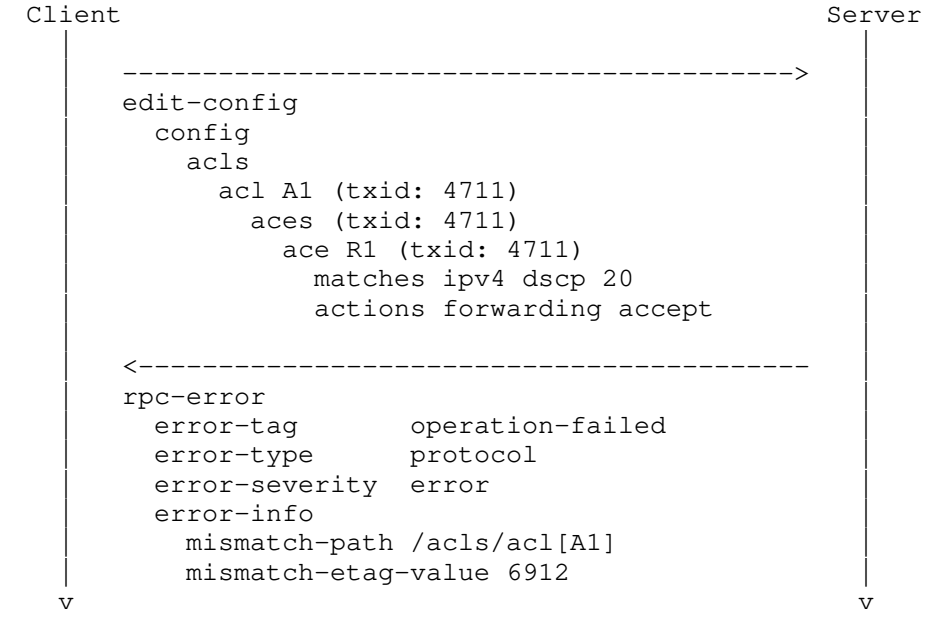


Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the c-txid it knows for this part of the configuration. Since the s-txid has changed (out of band), the server rejects the configuration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

3.6.2. Txid History size consideration

It may be tempting for a client implementor to send only the top level c-txid value for the tree being edited. In most cases, that would certainly work just fine. This is a way for the client to request the server to go ahead with the change as long as there has not been any changes more recent than the client provided c-txid.

Here the client is sending the same change as in the example above (Figure 5), but with only one top level c-txid value.

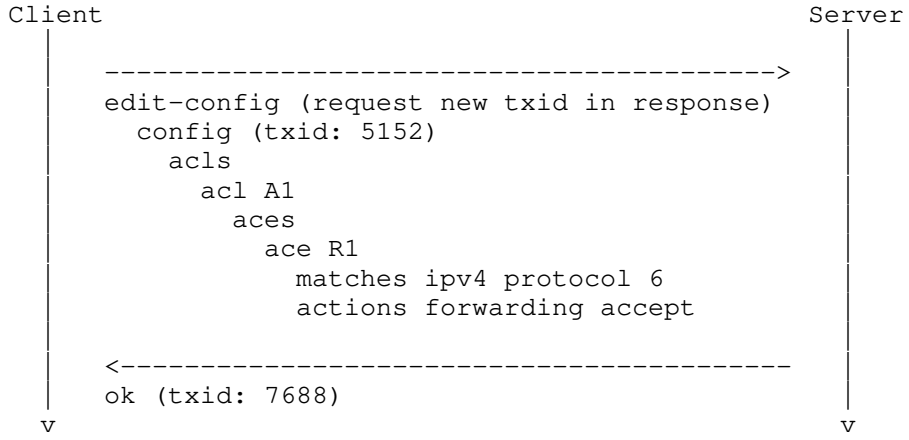


Figure 8: Conditional transaction towards the Running datastore successfully executed. As all the c-txid values specified by the client were the same or more recent in the server's Txid History, so the transaction was successfully executed.

This approach works well because the top level value is inherited down in the child nodes and the server finds this value to either match exactly or be a more recent s-txid value in the server's Txid History.

The only caveat is that by relying on the server's Txid History being long enough, the change could be rejected if the top level c-txid has fallen out of the server's Txid History. Some servers may have a Txid History size of zero. A client specifying a single top-level c-txid value towards such a server would not be able to get the transaction accepted.

3.7. Candidate Datastore Transactions

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

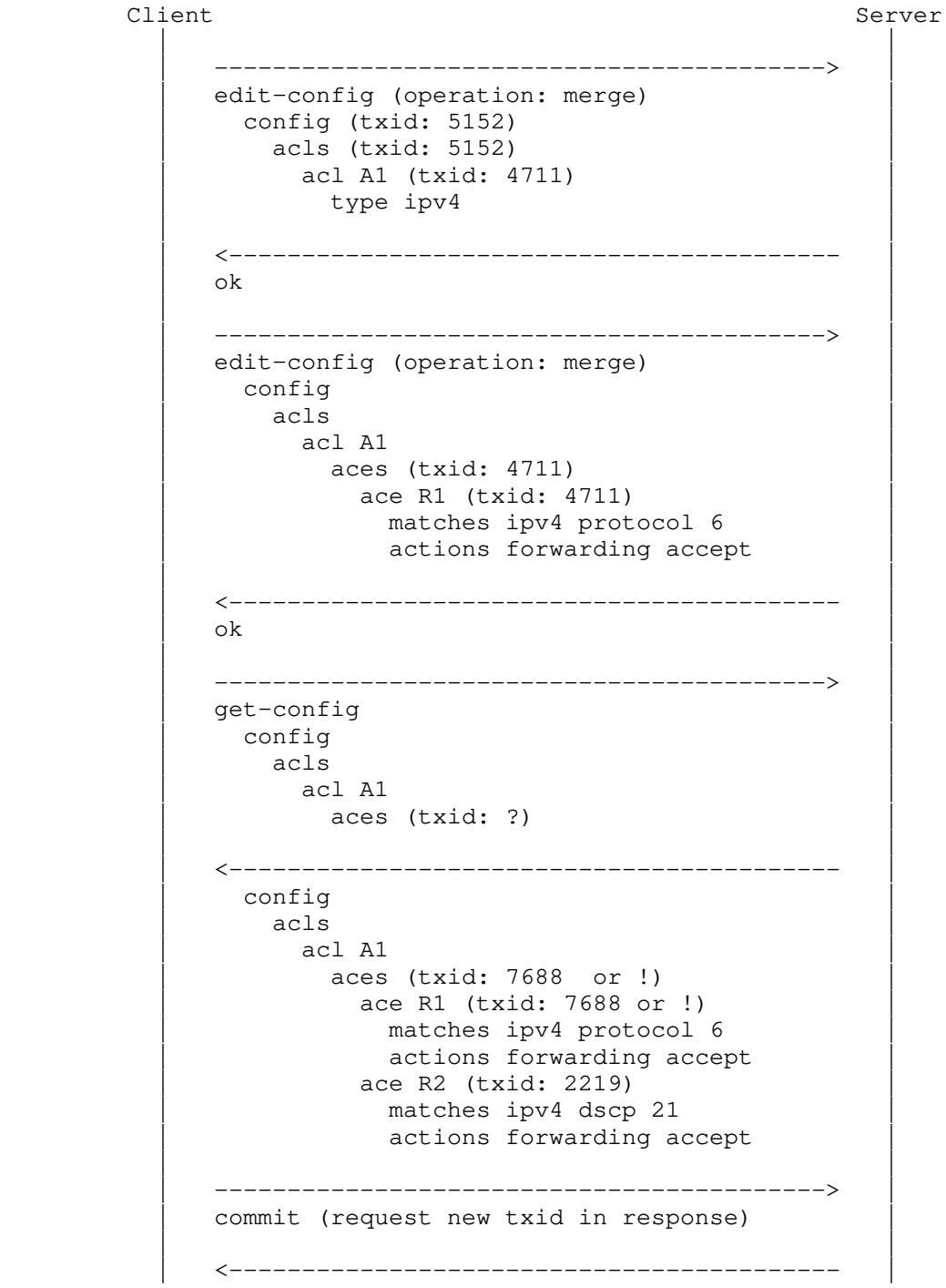

Client
Server
edit-config (operation: merge)
config (txid: 5152)
acls (txid: 5152)
acl A1 (txid: 4711)
type ipv4
ok
edit-config (operation: merge)
config
acls
acl A1
aces (txid: 4711)
ace R1 (txid: 4711)
matches ipv4 protocol 6
actions forwarding accept
ok
get-config
config
acls
acl A1
aces (txid: ?)
config
acls
acl A1
aces (txid: 7688 or !)
ace R1 (txid: 7688 or !)
matches ipv4 protocol 6
actions forwarding accept
ace R2 (txid: 2219)
matches ipv4 dscp 21
actions forwarding accept
commit (request new txid in response)

Figure 9: Conditional transaction towards the Candidate datastore successfully executed. As all the c-txid values specified by the client matched those on the server at the time of the commit, the transaction was successfully executed. If a client issues a get-config towards the candidate datastore, the server may choose to return the special txid-unknown value (e.g. "!") or the s-txid value that would be used if the candidate was committed without further changes (when that s-txid value is known in advance by the server).

3.8. Dependencies within Transactions

YANG modules that contain when-statements referencing remote parts of the model will cause the s-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:

```
module energy-example {
...

  container energy {
    leaf metering-enabled {
      type boolean;
      default false;
    }
  }

  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:

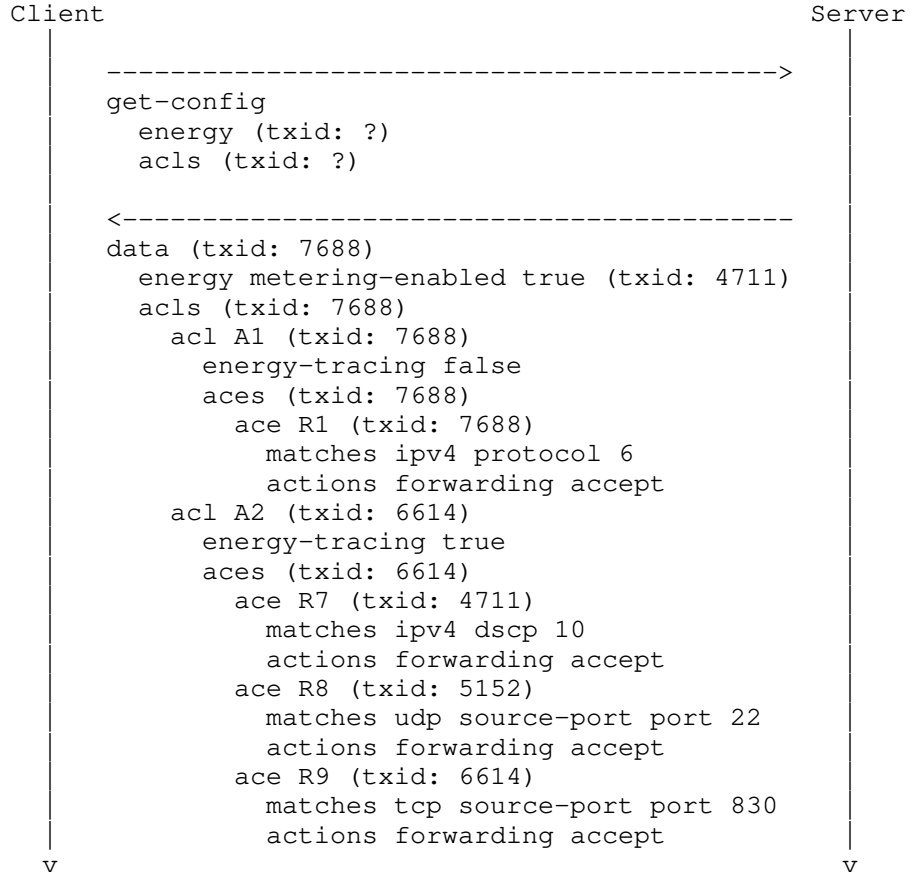


Figure 10: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

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 s-txid MUST be updated appropriately.

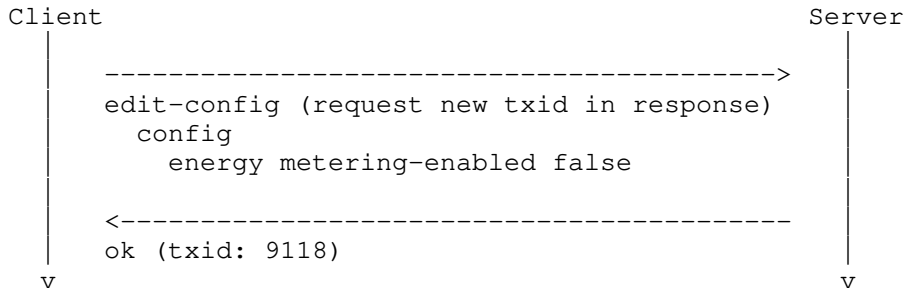


Figure 11: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in s-txid changes.

After the transaction above, the new configuration state has the energy-tracing leafs removed. Every such removal or (re)introduction of a node counts as a configuration change from a txid perspective, regardless of whether the change has any net configuration change effect in the server.

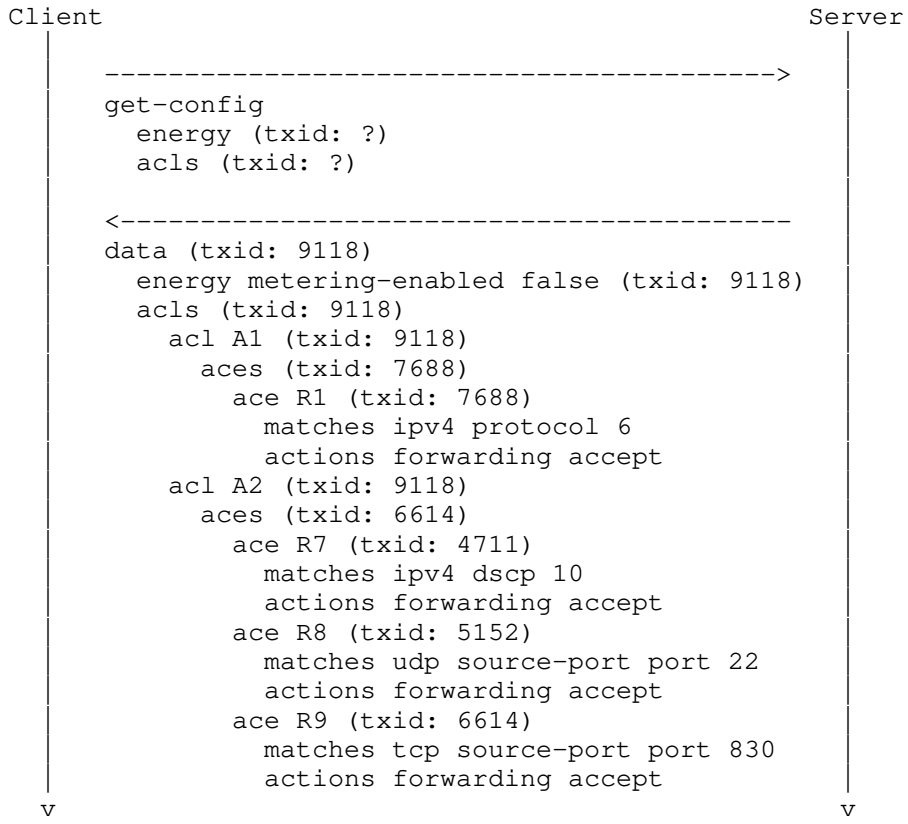


Figure 12: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when- expression. Both acl A1 and acl A2 have their txids updated, even though energy-tracing was already false for acl A1.

3.9. 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 are retained and changed as if the data being copied had been sent in through an edit-config 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 3.6) MUST be sent.

3.10. YANG-Push Subscriptions

A client issuing a YANG-Push establish-subscription or modify-subscription request towards a server that supports ietf-netconf-txid-yang-push.yang MAY request that the server provides updated txid values in YANG-Push on-change subscription updates.

This functionality pertains only to on-change updates. This RPC may also be invoked over RESTCONF or other protocols, and might therefore be encoded in JSON.

To request txid values (e.g. etag), the client adds a flag in the request (e.g. with-etag). The server then returns the txid (e.g. etag) value in the yang-patch payload (e.g. as etag-value).

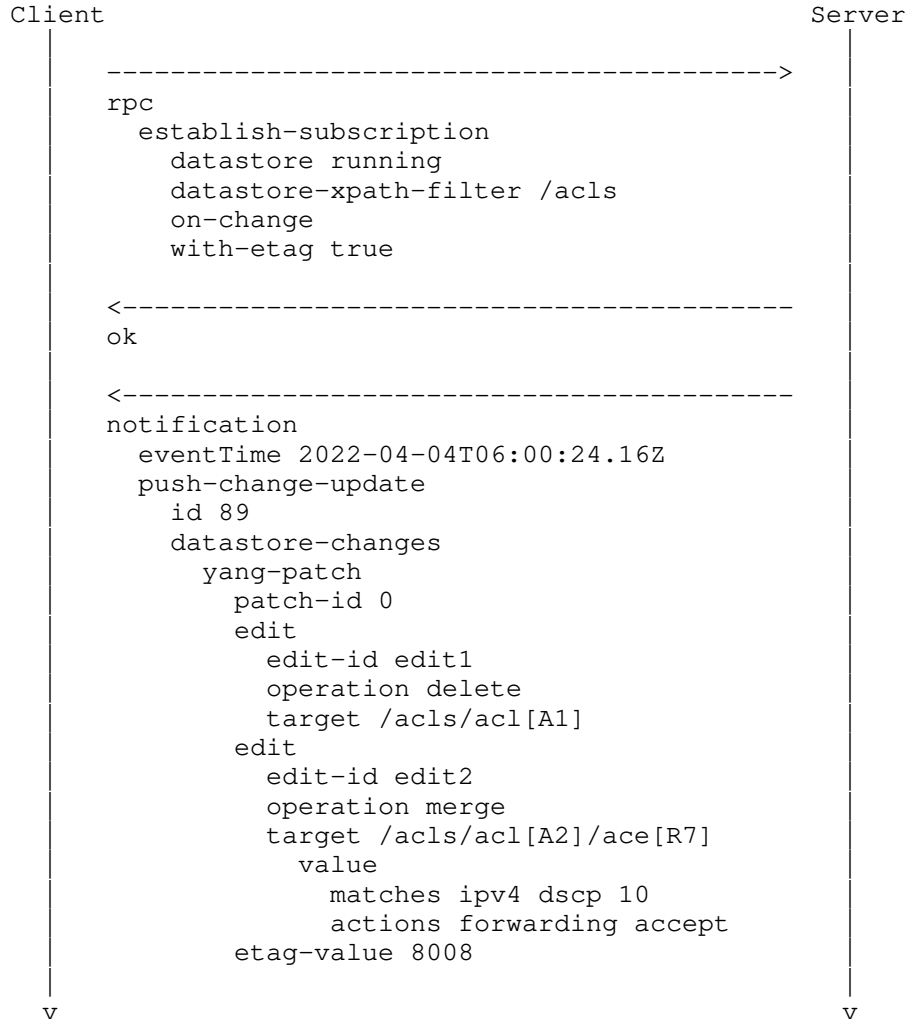


Figure 13: A client requests a YANG-Push subscription for a given path with txid value included. When the server delivers a push-change-update notification, the txid value pertaining to the entire patch is included.

3.11. Comparing YANG Datastores

A client issuing an NMDA Datastore compare request towards a server that supports ietf-netconf-txid-nmda-compare.yang MAY request that the server provides updated txid values in the compare reply. Besides NETCONF, this RPC may also be invoked over RESTCONF or other protocols, and might therefore be encoded in JSON.

To request txid values (e.g. etag), the client adds a flag in the request (e.g. with-etag). The server then returns the txid (e.g. etag) value in the yang-patch payload (e.g. as etag-value).

The txid value returned by the server MUST be the txid value pertaining to the target node in the source or target datastores that is the most recent. If one of the datastores being compared is not a configuration datastore, the txid in the configuration datastore MUST be used. If none of the datastores being compared are a configuration datastore, then txid values MUST NOT be returned at all.

The txid to return is the one that pertains to the target node, or in the case of delete, the closest surviving ancestor of the target node.

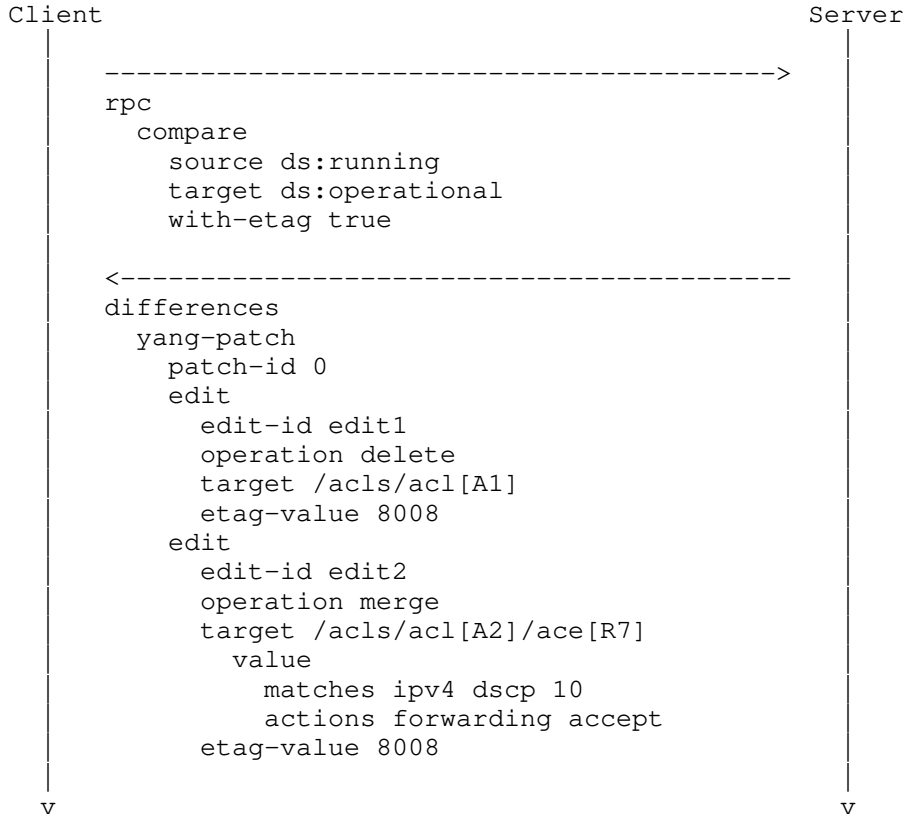


Figure 14: A client requests a NMDA Datastore compare for a given path with txid values included. When the server delivers the reply, the txid is included for each edit.

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 [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, [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 feature last-modified defined in ietf-netconf-txid.yang.

The last-modified attribute values are yang:date-and-time values as defined in ietf-yang-types.yang, [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 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 [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, [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 3.6) with an error-info tag containing a txid-value-mismatch-error-info structure.

4.3.1. Candidate Datastore

When servers return txid values in get-config and get-data operations towards the candidate datastore, the txid values returned MUST adhere to the following rules:

* If the versioned node holds the same data as in the running datastore, the same txid value as the versioned node in running MUST be used.

* If the versioned node is different in the candidate store than in the running datastore, the server has a choice of what to return. The server MAY return the special "txid-unknown" value "!". If the txid-unknown value is not returned, the server MUST return the txid value the versioned node will have if the client decides to commit the candidate datastore without further updates.

4.3.2. Namespaces and Attribute Placement

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.

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>17</protocol>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </ace>
          </aces>
        </acl>
        <acl txid:etag="nc5152">
          <name>A2</name>
          <aces txid:etag="nc5152">
            <ace txid:etag="nc4711">
              <name>R7</name>
              <matches>
                <ipv4>
                  <dscp>10</dscp>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </ace>
            <ace txid:etag="nc5152">
              <name>R8</name>
              <matches>
                <udp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </udp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
```

```
            </ace>
            <ace txid:etag="nc5152">
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </tcp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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>17</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
        </aces>
      </acl>
      <acl txid:etag="nc5152">
        <name>A2</name>
        <aces txid:etag="nc5152">
          <ace txid:etag="nc4711">
            <name>R7</name>
            <matches>
              <ipv4>
                <dscp>10</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <udp>
                <source-port>
                  <port>22</port>
                </source-port>
              </udp>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc5152">
```

```
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>22</port>
                  </source-port>
                </tcp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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>
        <aces txid:last-modified="2022-03-20T16:20:11.333444Z">
          <ace txid:last-modified="2022-03-20T16:20:11.333444Z">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>17</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
        </aces>
      </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>10</dscp>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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>10</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc5152">
            <name>R8</name>
            <matches>
              <ipv4>
                <source-port>
                  <port>22</port>
                </source-port>
              </ipv4>
```

```
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </ace>
            <ace txid:etag="nc6614">
              <name>R9</name>
              <matches>
                <ipv4>
                  <source-port>
                    <port>830</port>
                  </source-port>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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 (i.e. is identical to the server's txid value, or found earlier in the server's Txid History), 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>6</protocol>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                <forwarding>
              </actions>
            </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>6</protocol>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 <forwarding>
               </actions>
             </ace>
           </aces>
         </acl>
         <acl txid:etag="nc6614">
           <name>A2</name>
           <aces txid:etag="nc6614">
             <ace txid:etag="nc4711">
               <name>R7</name>
               <matches>
                 <ipv4>
                   <dscp>10</dscp>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 <forwarding>
               </actions>
             </ace>
             <ace txid:etag="nc5152">
               <name>R8</name>
               <matches>
                 <udp>
                   <source-port>
```

```
                    <port>22</port>
                  </source-port>
                </udp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </ace>
            <ace txid:etag="nc6614">
              <name>R9</name>
              <matches>
                <tcp>
                  <source-port>
                    <port>830</port>
                  </source-port>
                </tcp>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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>6</protocol>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </ace>
          </aces>
        </acl>
        <acl txid:etag="cli2222">
          <name>A2</name>
          <aces txid:etag="cli2222">
            <ace txid:etag="nc4711">
              <name>R7</name>
              <matches>
                <ipv4>
                  <dscp>10</dscp>
                </ipv4>
              </matches>
              <actions>
                <forwarding xmlns:acl=
                "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                  acl:accept
                <forwarding>
              </actions>
            </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>
      <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">
        <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>10</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </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, nor was the client's txid value found later in the server's Txid History, then the server 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. Reading from the Candidate Datastore

Let’s assume that a get-config towards the running datastore currently contains the following data and txid values:

```
   <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">
     <data>
       <acls
         xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
         txid:etag="nc4711">
         <acl txid:etag="nc4711">
           <name>A1</name>
           <aces txid:etag="nc4711">
             <ace txid:etag="nc4711">
               <name>R1</name>
               <matches>
                 <ipv4>
                   <protocol>17</protocol>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 <forwarding>
               </actions>
             </ace>
             <ace txid:etag="nc2219">
               <name>R2</name>
               <matches>
                 <ipv4>
                   <dscp>21</dscp>
                 </ipv4>
               </matches>
               <actions>
                 <forwarding xmlns:acl=
                 "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                   acl:accept
                 <forwarding>
               </actions>
             </ace>
           </aces>
         </acl>
       </acls>
     </data>
   </rpc-reply>
```

A client issues discard-changes (to make the candidate datastore equal to the running datastore), and issues an edit-config to change the R1 protocol from udp (17) to tcp (6), and then executes a get-config with the txid-request attribute "?" set on the acl A1, the server might respond:

```
<rpc-reply message-id="13"
           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 txid:etag="!">
        <name>A1</name>
        <aces txid:etag="!">
          <ace txid:etag="!">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>6</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc2219">
            <name>R2</name>
            <matches>
              <ipv4>
                <dscp>21</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

Here, the txid-unknown value "!" is sent by the server. This particular server implementation does not know beforehand which txid value would be used for this versioned node after commit. It will be a value different from the current corresponding txid value in the running datastore.

In case the server is able to predict the txid value that would be used for the versioned node after commit, it could respond with that value instead. Let's say the server knows the txid would be "7688" if the candidate datastore was committed without further changes, then it would respond with that value in each place where the example shows "!" above.

5.6. Commit

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="14"
    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="15"
    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>
```

5.7. 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="16"
             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:on-change/>
    <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="17"
    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"
  xmlns:ietf-netconf-txid-yp=
    "urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push">
  <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>
          <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>
        <ietf-netconf-txid-yp:etag-value>
          nc8008
        </ietf-netconf-txid-yp:etag-value>
      </yang-patch>
    </datastore-changes>
  </push-change-update>
</notification>
```

5.8. NMDA Compare

The following example is taken from section 5 of [RFC9144]. It compares the difference between the operational and intended datastores for a subtree under "interfaces".

In this version of the example, the client requests that txid values, in this case etag-values, are annotated to the result.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <compare xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-compare"
      xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores"
      xmlns:ietf-netconf-txid-nmda-compare=
        "urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare">
    <source>ds:operational</source>
    <target>ds:intended</target>
    <report-origin/>
    <ietf-netconf-txid-nmda-compare:with-etag>
      true
    </ietf-netconf-txid-nmda-compare:with-etag>
    <xpath-filter
        xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      /if:interfaces
    </xpath-filter>
  </compare>
</rpc>
```

RPC reply when a difference is detected:

```
<rpc-reply
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    message-id="101">
  <differences
    xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-compare"
    xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
    xmlns:ietf-netconf-txid-nmda-compare=
      "urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare">
    <yang-patch>
      <patch-id>interface status</patch-id>
      <comment>
        diff between operational (source) and intended (target),
        with txid values taken from intended.
      </comment>
      <edit>
        <edit-id>1</edit-id>
        <operation>replace</operation>
        <target>/ietf-interfaces:interface=eth0/enabled</target>
        <value>
          <if:enabled>false</if:enabled>
        </value>
        <source-value>
          <if:enabled or:origin="or:learned">true</if:enabled>
        </source-value>
        <ietf-netconf-txid-nmda-compare:etag-value>
          4004
        </ietf-netconf-txid-nmda-compare:etag-value>
      </edit>
      <edit>
        <edit-id>2</edit-id>
        <operation>create</operation>
        <target>/ietf-interfaces:interface=eth0/description</target>
        <value>
          <if:description>ip interface</if:description>
        </value>
        <ietf-netconf-txid-nmda-compare:etag-value>
          8008
        </ietf-netconf-txid-nmda-compare:etag-value>
      </edit>
    </yang-patch>
  </differences>
</rpc-reply>
```

The same response in RESTCONF (using JSON format):

```
   HTTP/1.1 200 OK
   Date: Thu, 24 Jan 2019 20:56:30 GMT
   Server: example-server
   Content-Type: application/yang-data+json

   { "ietf-nmda-compare:output" : {
       "differences" : {
         "ietf-yang-patch:yang-patch" : {
           "patch-id" : "interface status",
           "comment" : "diff between intended (source) and operational",
           "edit" : [
             {
               "edit-id" : "1",
               "operation" : "replace",
               "target" : "/ietf-interfaces:interface=eth0/enabled",
               "value" : {
                 "ietf-interfaces:interface/enabled" : "false"
               },
               "source-value" : {
                 "ietf-interfaces:interface/enabled" : "true",
                 "@ietf-interfaces:interface/enabled" : {
                   "ietf-origin:origin" : "ietf-origin:learned"
                 }
               },
               "ietf-netconf-txid-nmda-compare:etag-value": "4004"
             },
             {
               "edit-id" : "2",
               "operation" : "create",
               "target" : "/ietf-interfaces:interface=eth0/description",
               "value" : {
                 "ietf-interface:interface/description" : "ip interface"
               },
               "ietf-netconf-txid-nmda-compare:etag-value": "8008"
             }
           ]
         }
       }
     }
   }
```

6. YANG Modules

6.1. Base module for txid in NETCONF

```
<CODE BEGINS>
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 2023-03-01 {
     description
       "Initial revision";
     reference
       "RFC XXXX: Xxxxxxxxx";
   }

   feature last-modified {
     description "Servers implementing this module MUST support the
       etag txid mechanism.  Servers MAY also support the
       last-modified txid mechanism.  Support is shown by announcing
       this feature.";
   }

   extension versioned-node {
     description "This statement is used by servers to declare that a
       the server is maintaining a Txid for the YANG node with this
       statement.  Which YANG nodes are versioned nodes may be useful
       information for clients (especially during development).

       Servers are not required to use this statement to declare which
       nodes are versioned nodes.

       Example of use:

       container interfaces {
         ietf-netconf-txid:versioned-node;
         ...
       }
       ";
   }

   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
              the node in the candidate datastore has changed
              relative the running datastore, but not yet received
              a new txid value 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 {
        if-feature 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.";
    }

    grouping txid-value-grouping {
      leaf etag-value {
        type etag-t;
        description
          "Indicates server's txid value for a YANG node.";
      }
      leaf last-modified-value {
        if-feature last-modified;
        type last-modified-t;
        description
          "Indicates server's txid value for a YANG node.";
      }
      description
        "Grouping for txid mechanisms, to be augmented into
         output of rpcs that return txid metadata for 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 {
          if-feature last-modified;
          type last-modified-t;
          description
            "Indicates server's txid value of the last-modified
            attribute for one mismatching element.";
        }
      }
    }
  }
  <CODE ENDS>
```

6.2. Additional support for txid in YANG-Push

```
  <CODE BEGINS>
  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-yang-push {
    prefix yp;
    reference
      "RFC 8641: Subscriptions to YANG Datastores";
  }

  import ietf-yang-patch {
    prefix ypatch;
    reference
      "RFC 8072: YANG Patch Media Type";
  }

  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;
    }
    augment "/yp:push-change-update/yp:datastore-changes/" +
            "yp:yang-patch" {
      description
        "This augmentation makes it possible for servers to return
        txid-values.";
      uses ietf-netconf-txid:txid-value-grouping;
    }
  }
  <CODE ENDS>
```

6.3. Additional support for txid in NMDA Compare

```
<CODE BEGINS>
module ietf-netconf-txid-nmda-compare {
  yang-version 1.1;
  namespace
    'urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare';
  prefix ietf-netconf-txid-nmda-compare;

  import ietf-nmda-compare {
    prefix cmp;
    reference
      "RFC 9144: Comparison of Network Management Datastore
       Architecture (NMDA) Datastores";
  }

  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 NMDA Compare.

     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 2023-05-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Xxxxxxxxx";
    }

    augment "/cmp:compare/cmp:input" {
      description
        "This augmentation makes it possible for clients to request
         txids to be returned.";
      uses ietf-netconf-txid:txid-grouping;
    }
    augment "/cmp:compare/cmp:output/cmp:compare-response/" +
            "cmp:differences/cmp:differences/cmp:yang-patch/cmp:edit" {
      description
        "This augmentation makes it possible for servers to return
        txid-values.";
      container most-recent {
        uses ietf-netconf-txid:txid-value-grouping;
      }
    }
  }
  <CODE ENDS>
```

7. Security Considerations

7.1. NACM Access Control

NACM, [RFC8341], access control processing happens as usual, independently of any txid handling, if supported by the server and enabled by the NACM configuration.

It should be pointed out however, that when txid information is added to a reply, it may occasionally be possible for a client to deduce that a configuration change has happened in some part of the configuration to which it has no access rights.

For example, a client may notice that the root node txid has changed while none of the subtrees it has access to have changed, and thereby conclude that someone else has made a change to some part of the configuration that is not acessible by the client.

7.1.1. Hash-based Txid Algorithms

Servers that implement NACM and choose to implement a hash-based txid algorithm over the configuration may reveal to a client that the configuration of a subtree that the client has no access to is the same as it was at an earlier point in time.

For example, a client with partial access to the configuration might observe that the root node txid was 1234. After a few configuration changes by other parties, the client may again observe that the root node txid is 1234. It may then deduce that the configuration is the same as earlier, even in the parts of the configuration it has no access to.

In some use cases, this behavior may be considered a feature, since it allows a security client to verify that the configuration is the same as expected, without transmitting or storing the actual configuration.

7.2. Unchanged Configuration

It will also be possible for clients to deduce that a configuration change has not happened during some period, by simply observing that the root node (or other subtree) txid remains unchanged. This is true regardless of NACM being deployed or choice of txid algorithm.

Again, there may be use cases where this behavior may be considered a feature, since it allows a security client to verify that the configuration is the same as expected, without transmitting or storing the actual configuration.

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 four XML namespace URNs in the 'IETF XML registry', following the format defined in [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

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

Registrant Contact: The NETCONF WG of the IETF.

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

This document registers three module names in the 'YANG Module Names' registry, defined in [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

and

name: ietf-netconf-txid-nmda-compare

prefix: ietf-netconf-txid-nmda-compare

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

RFC: XXXX

9. Changes

9.1. Major changes in -02 since -01

* Added optional to implement Txid History concept in order to make the algorithm both more efficient and less verbose. Servers may still choose a Txid History size of zero, which makes the server behavior the same as in earlier versions of this document. Implementations that use txids consisting of a monotonically increasing integer or timestamp will be able to determine the sequnce of transactions in the history directly, making this trivially simple to implement.

* Added extension statement versioned-node, which servers may use to declare which YANG tree nodes are Versioned Nodes. This is entirely optional, however, but possibly useful to client developers.

* Renamed YANG feature ietf-netconf-txid:txid-last-modified to ietf-netconf-txid:last-modified in order to reduce redundant mentions of "txid".

9.2. Major changes in -01 since -00

* Changed YANG-push txid mechanism to use a simple leaf rather than an attribute to convey txid information. This is preferable since YANG-push content may be requested using other protocols than NETCONF and other encodings than XML. By removing the need for XML attributes in this context, the mechanism becomes significantly more portable.

* Added a section and YANG module augmenting the RFC9144 NMDA datastore compare operation to allow request and reply with txid information. This too is done with augments of plain leafs for maximum portability.

* Added note clarifying that the txid attributes used in the XML encoding are never used in JSON (since RESTCONF uses HTTP headers instead).

* Added note clarifying that pruning happens when client and server txids _match_, since the server sending information to the client only makes sense when the information on the client is out of date.

* Added note clarifying that this entire document is about config true data only.

* Rephrased slightly when referring to the candidate datastore to keep making sense in the event that private candidate datastores become a reality in the future.

* Added a note early on to more clearly lay out the structure of this document, with a first part about the generic mechanism part, and a second part about the two specific txid mechanisms.

* Corrected acl data model examples to conform to their YANG module.

9.3. Major changes in draft-ietf-netconf-transaction-id-00 since -02

* Changed the logic around how txids are handled in the candidate datastore, both when reading (get-config, get-data) and writing (edit-config, edit-data). Introduced a special "txid-unknown" value "!".

* Changed the logic of copy-config to be similar to edit-config.

* Clarified how txid values interact with when-dependencies together with default values.

* Added content to security considerations.

* Added a high-level example for YANG-Push subscriptions with txid.

* Updated language about error-info sent at txid mismatch in an edit-config: error-info with mismatch details MUST be sent when mismatch detected, and that the server can choose one of the txid mismatch occurrences if there is more than one.

* Some rewording and minor additions for clarification, based on mailing list feedback.

* Divided RFC references into normative and informative.

* Corrected a logic error in the second figure (figure 6) in the "Conditional Transactions" section

9.4. 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 [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.

9.5. 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. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/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/rfc/rfc6241>.

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

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

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

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

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

[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/rfc/rfc9144>.

10.2. Informative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/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/rfc/rfc6020>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/rfc/rfc7232>.

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

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

Acknowledgments

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

Author's Address

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 13 April 2024

G. Zheng
T. Zhou
Huawei
T. Graf
Swisscom
P. Francois
A. Huang Feng
INSA-Lyon
P. Lucente
NTT

11 October 2023

UDP-based Transport for Configured Subscriptions
draft-ietf-netconf-udp-notif-11

Abstract

This document describes a UDP-based protocol for YANG notifications to collect data from network nodes. A shim header is proposed to facilitate the data streaming directly from the publishing process on network processor of line cards to receivers. The objective is to provide a lightweight approach to enable higher frequency and less performance impact on publisher and receiver processes compared to already established notification mechanisms.

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 13 April 2024.

Table of Contents

1. Introduction

The mechanism to support a subscription of a continuous and customized stream of updates from a YANG datastore [RFC8342] is defined in [RFC8639] and [RFC8641] and is abbreviated as Sub-Notif. Requirements for Subscription to YANG Datastores are defined in [RFC7923].

The mechanism separates the management and control of subscriptions from the transport used to deliver the data. Three transport mechanisms, namely NETCONF transport [RFC8640], RESTCONF transport [RFC8650], and HTTPS transport [I-D.ietf-netconf-https-notif] have been defined so far for such notification messages.

While powerful in their features and general in their architecture, the currently available transport mechanisms need to be complemented to support data publications at high velocity from network nodes that feature a distributed architecture. The currently available transports are based on TCP and lack the efficiency needed to continuously send notifications at high velocity.

This document specifies a transport option for Sub-Notif that leverages UDP. Specifically, it facilitates the distributed data collection mechanism described in [I-D.ietf-netconf-distributed-notif]. In the case of publishing from multiple network processors on multiple line cards, centralized designs require data to be internally forwarded from those network processors to the push server, presumably on a route processor, which then combines the individual data items into a single consolidated stream. The centralized data collection mechanism can result in a performance bottleneck, especially when large amounts of data are involved.

What is needed is a mechanism that allows for directly publishing from multiple network processors on line cards, without passing them through an additional processing stage for internal consolidation. The proposed UDP-based transport allows for such a distributed data publishing approach.

* Firstly, a UDP approach reduces the burden of maintaining a large amount of active TCP connections at the receiver, notably in cases where it collects data from network processors on line cards from a large amount of network nodes.

* Secondly, as no connection state needs to be maintained, UDP encapsulation can be easily implemented by the hardware of the publication streamer, which further improves performance.

* Ultimately, such advantages allow for a larger data analysis feature set, as more voluminous, finer grained data sets can be streamed to the receiver.

The transport described in this document can be used for transmitting notification messages over both IPv4 and IPv6.

This document describes the notification mechanism. It is intended to be used in conjunction with [RFC8639], extended by [I-D.ietf-netconf-distributed-notif].

Section 2 describes the control of the proposed transport mechanism. Section 3 details the notification mechanism and message format. Section 4 describes the use of options in the notification message header. Section 5 covers the applicability of the proposed mechanism. Section 6 describes a mechanism to secure the protocol in open networks.

2. Configured Subscription to UDP-Notif

This section describes how the proposed mechanism can be controlled using subscription channels based on NETCONF or RESTCONF.

As specified in Sub-Notif, configured subscriptions contain the location information of all the receivers, including the IP address and the port number, so that the publisher can actively send UDP-Notif messages to the corresponding receivers.

Note that receivers MAY NOT be already up and running when the configuration of the subscription takes effect on the monitored network node. The first message MUST be a separate subscription-started notification to indicate the Receiver that the stream has started flowing. Then, the notifications can be sent immediately

without delay. All the subscription state notifications, as defined in Section 2.7 of [RFC8639], MUST be encapsulated in separate notification messages.

3. UDP-Based Transport

In this section, we specify the UDP-Notif Transport behavior. Section 3.1 describes the general design of the solution. Section 3.2 specifies the UDP-Notif message format and Section 3.3 describes the encoding of the message payload.

3.1. Design Overview

As specified in Sub-Notif, the YANG data is encapsulated in a NETCONF/RESTCONF notification message, which is then encapsulated and carried using a transport protocols such as TLS or HTTP2. This document defines a UDP based transport. Figure 1 illustrates the structure of an UDP-Notif message.

* The Message Header contains information that facilitate the message transmission before deserializing the notification message.

* Notification Message is the encoded content that is transported by the publication stream. The common encoding methods are listed in Section 3.2. The structure of the Notification Message is defined in Section 2.6 of [RFC8639] and a YANG model has been proposed in [I-D.ahuang-netconf-notif-yang]. [I-D.ietf-netconf-notification-messages] proposes a structure to send bundled notifications in a single message.

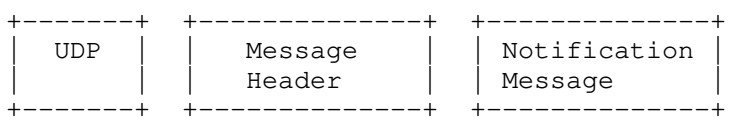


Figure 1: UDP-Notif Message Overview

3.2. Format of the UDP-Notif Message Header

The UDP-Notif Message Header contains information that facilitate the message transmission before deserializing the notification message. The data format is shown in Figure 2.

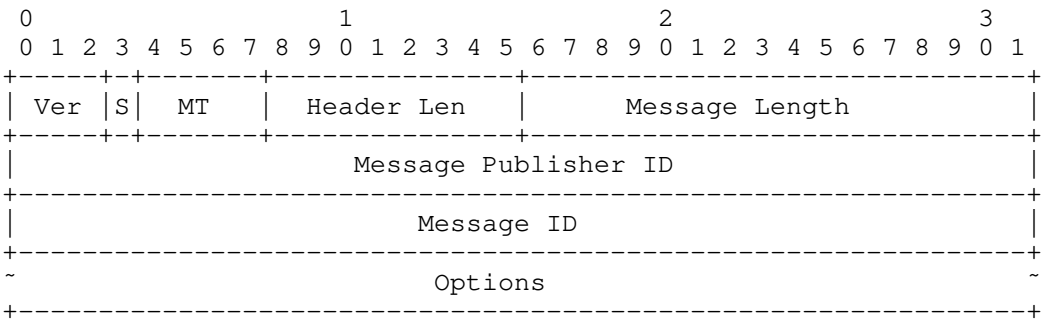


Figure 2: UDP-Notif Message Header Format

The Message Header contains the following field:

* Ver indicates the UDP-notif protocol header version. The values are allocated by the IANA registry "UDP-notif header version". The current header version number is 1.

* S represents the space of media type specified in the MT field. When S is unset, MT represents the standard media types as defined in this document. When S is set, MT represents a private space to be freely used for non standard encodings.

* MT is a 4 bit identifier to indicate the media type used for the Notification Message. 16 types of encoding can be expressed. When the S bit is unset, the following values apply:

  - 0: Reserved;

  - 1: application/yang-data+json [RFC8040]

  - 2: application/yang-data+xml [RFC8040]

  - 3: application/yang-data+cbor [RFC9254]

* Header Len is the length of the message header in octets, including both the fixed header and the options.

* Message Length is the total length of the UDP-notif message within one UDP datagram, measured in octets, including the message header. When the Notification Message is segmented using the Segmentation Options defined in Section 4.1 the Message Length is the total length of the current, segmented UDP-notif message, not the length of the entire Notification message.

* Message Publisher ID is a 32-bit identifier defined in [I-D.ietf-netconf-distributed-notif]. This identifier is unique to the publisher node and identifies the publishing process of the node to allow the disambiguation of an information source. Message unicity is obtained from the conjunction of the Message Publisher ID and the Message ID field described below. If Message Publisher ID unicity is not preserved through the collection domain, the source IP address of the UDP datagram SHOULD be used in addition to the Message Publisher ID to identify the information source. If a transport layer relay is used, Message Publisher ID unicity must be preserved through the collection domain.

* The Message ID is generated continuously by the publisher of UDP-Notif messages. A publisher MUST use different Message ID values for different messages generated with the same Message Publisher ID. Note that the main purpose of the Message ID is to reconstruct messages which were segmented using the segmentation option described in section Section 4.1. The Message ID values SHOULD be incremented by one for each successive message originated with the same Message Publisher ID, so that message loss can be detected. Furthermore, incrementing the Message ID by one allows for a large amount of time to happen before the Message ID's are reused due to wrapping around. Different subscribers MAY share the same Message ID sequence.

* Options is a variable-length field in the TLV format. When the Header Length is larger than 12 octets, which is the length of the fixed header, Options TLVs follow directly after the fixed message header (i.e., Message ID). The details of the options are described in Section 4.

3.3. Data Encoding

UDP-Notif message data can be encoded in CBOR, XML or JSON format. It is conceivable that additional encodings may be supported in the future. This can be accomplished by augmenting the subscription data model with additional identity statements used to refer to requested encodings.

Private encodings can be using the S bit of the header. When the S bit is set, the value of the MT field is left to be defined and agreed upon by the users of the private encoding. An option is defined in Section 4.2 for more verbose encoding descriptions than what can be described with the MT field.

Implementation MAY support multiple encoding methods per subscription. When bundled notifications are supported between the publisher and the receiver, only subscribed notifications with the same encoding can be bundled in a given message.

4. Options

All the options are defined with the following format, illustrated in Figure 3.

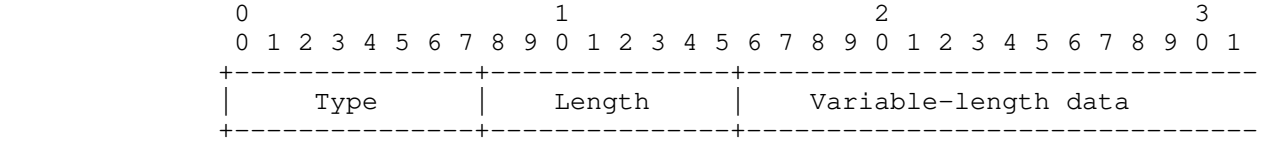


Figure 3: Generic Option Format

* Type: 1 octet describing the option type;

* Length: 1 octet representing the total number of octets in the TLV, including the Type and Length fields;

* Variable-length data: 0 or more octets of TLV Value.

When more than one option is used in the UDP-notif header, options MUST be ordered by the Type value. Messages with unordered options MAY be dropped by the Receiver.

4.1. Segmentation Option

The UDP payload length is limited to 65535. Application level headers will make the actual payload shorter. Even though binary encodings such as CBOR may not require more space than what is left, more voluminous encodings such as JSON and XML may suffer from this size limitation. Although IPv4 and IPv6 publishers can fragment outgoing packets exceeding their Maximum Transmission Unit (MTU), fragmented IP packets may not be desired for operational and performance reasons.

Consequently, implementations of the mechanism SHOULD provide a configurable max-segment-size option to control the maximum size of a payload.

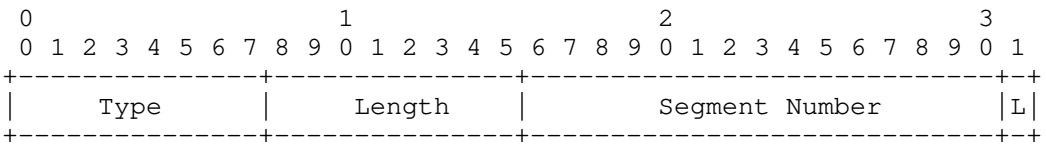


Figure 4: Segmentation Option Format

The Segmentation Option is to be included when the message content is segmented into multiple segments. Different segments of one message share the same Message ID. An illustration is provided in Figure 4. The fields of this TLV are:

* Type: Generic option field which indicates a Segmentation Option. The Type value is to be assigned TBD1.

* Length: Generic option field which indicates the length of this option. It is a fixed value of 4 octets for the Segmentation Option.

* Segment Number: 15-bit value indicating the sequence number of the current segment. The first segment of a segmented message has a Segment Number value of 0.

* L: is a flag to indicate whether the current segment is the last one of the message. When 0 is set, the current segment is not the last one. When 1 is set, the current segment is the last one, meaning that the total number of segments used to transport this message is the value of the current Segment Number + 1.

An implementation of this specification SHOULD NOT rely on IP fragmentation by default to carry large messages. An implementation of this specification SHOULD either restrict the size of individual messages carried over this protocol, or support the segmentation option.

When a message has multiple options and is segmented using the described mechanism, all the options MUST be present on the first segment ordered by the options Type. The rest of segmented messages MAY include all the options ordered by options type.

4.2. Private Encoding Option

The space to describe private encodings in the MT field of the UDP-Notif header being limited, an option is provided to describe custom encodings. The fields of this option are as follows.

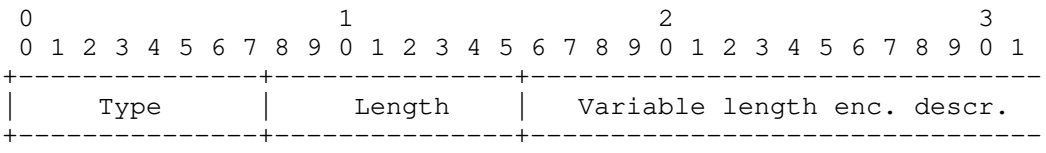


Figure 5: Private Encoding Option Format

* Type: Generic option field which indicates a Private Encoding Option. The Type value is to be assigned TBD2.

* Length: Generic option field which indicates the length of this option. It is a variable value.

* Enc. Descr: The description of the private encoding used for this message. The values to be used for such private encodings is left to be defined by the users of private encodings.

This option SHOULD only be used when the S bit of the header is set, as providing a private encoding description for standard encodings is meaningless.

5. Applicability

In this section, we provide an applicability statement for the proposed mechanism, following the recommendations of [RFC8085].

The proposed mechanism falls in the category of UDP applications "designed for use within the network of a single network operator or on networks of an adjacent set of cooperating network operators, to be deployed in controlled environments", as defined in [RFC8085]. Implementations of the proposed mechanism SHOULD thus follow the recommendations in place for such specific applications. In the following, we discuss recommendations on congestion control, message size guidelines, reliability considerations and security considerations.

The main use case of the proposed mechanism is the collection of statistical metrics for accounting purposes, where potential loss is not a concern, but should however be reported (such as IPFIX Flow Records exported with UDP [RFC7011]). Such metrics are typically exported in a periodical subscription as described in Section 3.1 of [RFC8641].

5.1. Congestion Control

The proposed application falls into the category of applications performing transfer of large amounts of data. It is expected that the operator using the solution configures QoS on its related flows. As per [RFC8085], such applications MAY choose not to implement any form of congestion control, but follow the following principles.

It is NOT RECOMMENDED to use the proposed mechanism over congestion-sensitive network paths. The only environments where UDP-Notif is expected to be used are managed networks. The deployments require that the network path has been explicitly provisioned to handle the traffic through traffic engineering mechanisms, such as rate limiting or capacity reservations.

Implementation of the proposal SHOULD NOT push unlimited amounts of traffic by default, and SHOULD require the users to explicitly configure such a mode of operation.

Burst mitigation through packet pacing is RECOMMENDED. Disabling burst mitigation SHOULD require the users to explicitly configure such a mode of operation.

Applications SHOULD monitor packet losses and provide means to the user for retrieving information on such losses. The UDP-Notif Message ID can be used to deduce congestion based on packet loss detection. Hence the receiver can notify the Publisher to use a lower streaming rate. The interaction to control the streaming rate on the Publisher is out of the scope of this document.

5.2. Message Size

[RFC8085] recommends not to rely on IP fragmentation for messages whose size result in IP packets exceeding the MTU along the path. The segmentation option of the current specification permits segmentation of the UDP Notif message content without relying on IP fragmentation. Implementation of the current specification SHOULD allow for the configuration of the MTU.

5.3. Reliability

A receiver implementation for this protocol SHOULD deal with potential loss of packets carrying a part of segmented payload, by discarding packets that were received, but cannot be re-assembled as a complete message within a given amount of time. This time SHOULD be configurable.

6. Secured layer for UDP-notif

In open or unsecured networks, UDP-notif messages MUST be secured or encrypted. In this section, a mechanism using DTLS 1.3 to secure UDP-notif protocol is presented. The following sections defines the requirements for the implementation of the secured layer of DTLS for UDP-notif. No DTLS 1.3 extensions are defined in this document.

The DTLS 1.3 protocol [RFC9147] is designed to meet the requirements of applications that need to secure datagram transport. Implementations using DTLS to secure UDP-notif messages MUST use DTLS 1.3 protocol as defined in [RFC9147].

When this security layer is used, the Publisher MUST always be a DTLS client, and the Receiver MUST always be a DTLS server. The Receivers MUST support accepting UDP-notif Messages on the specified UDP port, but MAY be configurable to listen on a different port. The Publisher MUST support sending UDP-notif messages to the specified UDP port, but MAY be configurable to send messages to a different port. The Publisher MAY use any source UDP port for transmitting messages.

6.1. Session lifecycle

6.1.1. DTLS Session Initiation

The Publisher initiates a DTLS connection by sending a DTLS ClientHello to the Receiver. Implementations MAY support the denial of service countermeasures defined by DTLS 1.3 if a given deployment can ensure that DoS attacks are not a concern. When these countermeasures are used, the Receiver responds with a DTLS HelloRetryRequest containing a stateless cookie. The Publisher sends a second DTLS ClientHello message containing the received cookie. Details can be found in Section 5.1 of [RFC9147].

When DTLS is implemented, the Publisher MUST NOT send any UDP-notif messages before the DTLS handshake has successfully completed. Early data mechanism (also known as 0-RTT data) as defined in [RFC9147] MUST NOT be used.

Implementations of this security layer MUST support DTLS 1.3 [RFC9147] and MUST support the mandatory to implement cipher suite TLS_AES_128_GCM_SHA256 and SHOULD implement TLS_AES_256_GCM_SHA384 and TLS_CHACHA20_POLY1305_SHA256 cipher suites, as specified in TLS 1.3 [RFC8446]. If additional cipher suites are supported, then implementations MUST NOT negotiate a cipher suite that employs NULL integrity or authentication algorithms.

Where confidentiality protection with DTLS is required, implementations must negotiate a cipher suite that employs a non-NULL encryption algorithm.

6.1.2. Publish Data

When DTLS is used, all UDP-notif messages MUST be published as DTLS "application_data". It is possible that multiple UDP-notif messages are contained in one DTLS record, or that a publication message is transferred in multiple DTLS records. The application data is defined with the following ABNF [RFC5234] expression:

```
APPLICATION-DATA = 1*UDP-NOTIF-FRAME

UDP-NOTIF-FRAME = MSG-LEN SP UDP-NOTIF-MSG

MSG-LEN = NONZERO-DIGIT *DIGIT

SP = %d32

NONZERO-DIGIT = %d49-57

DIGIT = %d48 / NONZERO-DIGIT
```

UDP-NOTIF-MSG is defined in Section 3.

The Publisher SHOULD attempt to avoid IP fragmentation by using the Segmentation Option in the UDP-notif message.

6.1.3. Session termination

A Publisher MUST close the associated DTLS connection if the connection is not expected to deliver any UDP-notif Messages later. It MUST send a DTLS close_notify alert before closing the connection. A Publisher (DTLS client) MAY choose to not wait for the Receiver's close_notify alert and simply close the DTLS connection. Once the Receiver gets a close_notify from the Publisher, it MUST reply with a close_notify.

When no data is received from a DTLS connection for a long time, the Receiver MAY close the connection. Implementations SHOULD set the timeout value to 10 minutes but application specific profiles MAY recommend shorter or longer values. The Receiver (DTLS server) MUST attempt to initiate an exchange of close_notify alerts with the Publisher before closing the connection. Receivers that are unprepared to receive any more data MAY close the connection after sending the close_notify alert.

Although closure alerts are a component of TLS and so of DTLS, they, like all alerts, are not retransmitted by DTLS and so may be lost over an unreliable network.

7. A YANG Data Model for Management of UDP-Notif

7.1. Generic grouping for UDP-based applications

The "ietf-udp-client" module defines a generic "grouping" to configure a UDP client.

7.1.1. YANG Tree

The following tree diagram [RFC8340] illustrates the "udp-client-grouping" grouping:

```
module: ietf-udp-client

  grouping udp-client-grouping:
    +-- remote-address    inet:ip-address-no-zone
    +-- remote-port       inet:port-number
    +-- dtls! {dtls13}?
       +-- client-identity!
       |  +-- (auth-type)
       |     +--:(certificate) {client-ident-x509-cert}?
       |     |  +-- certificate
       |     |        ...
       |     +--:(raw-public-key) {client-ident-raw-public-key}?
       |     |  +-- raw-private-key
       |     |        ...
       |     +--:(tls12-psk)
       |     |        {client-ident-tls12-psk,not tlsc:client-ident-tls12-psk}?
       |     |  +-- tls12-psk
       |     |        ...
       |     +--:(tls13-epsk) {client-ident-tls13-epsk}?
       |        +-- tls13-epsk
       |              ...
       +-- server-authentication
       |  +-- ca-certs! {server-auth-x509-cert}?
       |  |  +-- (local-or-truststore)
       |  |     +--:(local) {local-definitions-supported}?
       |  |     |     ...
       |  |     +--:(truststore)
       |  |              {central-truststore-supported,certificates}?
       |  |           ...
       |  +-- ee-certs! {server-auth-x509-cert}?
       |  |  +-- (local-or-truststore)
       |  |     +--:(local) {local-definitions-supported}?
```

```
      |  |     |     ...
      |  |     +--:(truststore)
      |  |              {central-truststore-supported,certificates}?
      |  |           ...
      |  +-- raw-public-keys! {server-auth-raw-public-key}?
      |  |  +-- (local-or-truststore)
      |  |     +--:(local) {local-definitions-supported}?
      |  |     |     ...
      |  |     +--:(truststore)
      |  |              {central-truststore-supported,public-keys}?
      |  |           ...
      |  +-- tls12-psks?        empty
      |  |       {server-auth-tls12-psk,not tlsc:server-auth-tls12-psk}?
      |  +-- tls13-epsks?       empty {server-auth-tls13-epsk}?
      +-- hello-params {tlscmn:hello-params}?
      |  +-- tls-versions
      |  |  +-- tls-version*   identityref
      |  +-- cipher-suites
      |     +-- cipher-suite*   identityref
      +-- keepalives {tls-client-keepalives}?
         +-- peer-allowed-to-send?   empty
         +-- test-peer-aliveness!
            +-- max-wait?       uint16
            +-- max-attempts?   uint8
```

7.1.2. YANG Module

The "ietf-udp-client" module defines a reusable "udp-client-grouping" grouping with the remote server IP address, remote port and a DTLS container to configure DTLS1.3 when DTLS encryption is supported. When configuring the DTLS layer, the grouping uses "tls-client-grouping" defined in [I-D.ietf-netconf-tls-client-server] to add DTLS 1.3 parameters.

```
<CODE BEGINS> file "ietf-udp-client@2023-05-08.yang"
module ietf-udp-client {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-udp-client";
  prefix udpc;
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-tls-client {
    prefix tlsc;
    reference
```

```
        "RFC TTTT: YANG Groupings for TLS Clients and TLS Servers";
    }

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

       Authors:  Alex Huang Feng
                 <mailto:alex.huang-feng@insa-lyon.fr>
                 Pierre Francois
                 <mailto:pierre.francois@insa-lyon.fr>
                 Guangying Zheng
                 <mailto:zhengguangying@huawei.com>
                 Tianran Zhou
                 <mailto:zhoutianran@huawei.com>
                 Thomas Graf
                 <mailto:thomas.graf@swisscom.com>
                 Paolo Lucente
                 <mailto:paolo@ntt.net>";

    description
      "Defines a generic grouping for UDP-based client applications.

      Copyright (c) 2023 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

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

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

    revision 2023-05-08 {
      description
        "Initial revision";
      reference
        "RFC-to-be: UDP-based Transport for Configured Subscriptions";
    }

   /*
    * FEATURES
    */
    feature dtls13 {
      description
```

```
        "This feature indicates that DTLS 1.3 encryption of UDP
         packets is supported.";
    }

    grouping udp-client-grouping {
      description
        "Provides a reusable grouping for configuring a UDP client.";

      leaf remote-address {
        type inet:ip-address-no-zone;
        mandatory true;
        description
          "IP address of the UDP client, which can be an
          IPv4 address or an IPV6 address.";
      }

      leaf remote-port {
        type inet:port-number;
        mandatory true;
        description
          "Port number of the UDP client.";
      }

      container dtls {
        if-feature dtls13;
        presence dtls;
        uses tlsc:tls-client-grouping {
          // Using tls-client-grouping without TLS1.2 parameters
          // allowing only DTLS 1.3
          refine "client-identity/auth-type/tls12-psk" {
            // create the logical impossibility of enabling TLS1.2
            if-feature "not tlsc:client-ident-tls12-psk";
          }
          refine "server-authentication/tls12-psks" {
            // create the logical impossibility of enabling TLS1.2
            if-feature "not tlsc:server-auth-tls12-psk";
          }
        }
        description
          "Container for configuring DTLS 1.3 parameters.";
      }
    }
  }
  <CODE ENDS>
```

7.2. YANG to configure UDP-notif

The YANG model described in Section 7.3 defines a new receiver instance for UDP-notif transport. When this transport is used, four new leaves and a dtls container allow configuring UDP-notif receiver parameters.

```
module: ietf-udp-notif-transport

  augment /sn:subscriptions/snr:receiver-instances
            /snr:receiver-instance/snr:transport-type:
    +--:(udp-notif)
       +--rw udp-notif-receiver
          +--rw remote-address          inet:ip-address-no-zone
          +--rw remote-port             inet:port-number
          +--rw dtls! {dtls13}?
          |  +--rw client-identity!
          |  |  +--rw (auth-type)
          |  |     +--:(certificate) {client-ident-x509-cert}?
          |  |     |     ...
          |  |     +--:(raw-public-key) {client-ident-raw-public-key}?
          |  |     |     ...
          |  |     +--:(tls13-epsk) {client-ident-tls13-epsk}?
          |  |           ...
          |  +--rw server-authentication
          |  |  +--rw ca-certs! {server-auth-x509-cert}?
          |  |  |  +--rw (local-or-truststore)
          |  |  |        ...
          |  |  +--rw ee-certs! {server-auth-x509-cert}?
          |  |  |  +--rw (local-or-truststore)
          |  |  |        ...
          |  |  +--rw raw-public-keys! {server-auth-raw-public-key}?
          |  |  |  +--rw (local-or-truststore)
          |  |  |        ...
          |  |  +--rw tls13-epsks?       empty
          |  |             {server-auth-tls13-epsk}?
          |  +--rw hello-params {tlscmn:hello-params}?
          |  |  +--rw tls-versions
          |  |  |  +--rw tls-version*   identityref
          |  |  +--rw cipher-suites
          |  |     +--rw cipher-suite*   identityref
          |  +--rw keepalives {tls-client-keepalives}?
          |     +--rw peer-allowed-to-send?   empty
          |     +--rw test-peer-aliveness!
          |        +--rw max-wait?       uint16
          |        +--rw max-attempts?   uint8
          +--rw enable-segmentation?   boolean {segmentation}?
          +--rw max-segment-size?      uint32 {segmentation}?
```

7.3. YANG Module

This YANG module is used to configure, on a publisher, a receiver willing to consume notification messages. This module augments the "ietf-subscribed-notif-receivers" module to define a UDP-notif transport receiver. The grouping "udp-notif-receiver-grouping" defines the necessary parameters to configure the transport defined in this document using the generic "udp-client-grouping" grouping.

```
<CODE BEGINS> file "ietf-udp-notif-transport@2023-05-08.yang"
module ietf-udp-notif-transport {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-udp-notif-transport";
  prefix unt;
  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-subscribed-notif-receivers {
    prefix snr;
    reference
      "RFC YYYY: An HTTPS-based Transport for
                 Configured Subscriptions";
  }
  import ietf-udp-client {
    prefix udpc;
    reference
      "RFC-to-be: UDP-based Transport for Configured Subscriptions";
  }

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

     Authors:  Guangying Zheng
               <mailto:zhengguangying@huawei.com>
               Tianran Zhou
               <mailto:zhoutianran@huawei.com>
               Thomas Graf
               <mailto:thomas.graf@swisscom.com>
               Pierre Francois
               <mailto:pierre.francois@insa-lyon.fr>
               Alex Huang Feng
               <mailto:alex.huang-feng@insa-lyon.fr>
               Paolo Lucente
```

```
                <mailto:paolo@ntt.net>";

     description
       "Defines UDP-Notif as a supported transport for subscribed
       event notifications.

       Copyright (c) 2023 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

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

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

     revision 2023-05-08 {
       description
         "Initial revision";
       reference
         "RFC-to-be: UDP-based Transport for Configured Subscriptions";
     }

    /*
     * FEATURES
     */
     feature encode-cbor {
       description
         "This feature indicates that CBOR encoding of notification
          messages is supported.";
     }
     feature segmentation {
       description
         "This feature indicates segmentation of notification messages
         is supported.";
     }

    /*
     * IDENTITIES
     */
     identity udp-notif {
       base sn:transport;
       description
         "UDP-Notif is used as transport for notification messages
           and state change notifications.";
     }
```

```
identity encode-cbor {
  base sn:encoding;
  description
    "Encode data using CBOR as described in RFC 9254.";
  reference
    "RFC 9254: CBOR Encoding of Data Modeled with YANG";
}

grouping udp-notif-receiver-grouping {
  description
    "Provides a reusable description of a UDP-Notif target
    receiver.";

  uses udpc:udp-client-grouping;

  leaf enable-segmentation {
    if-feature segmentation;
    type boolean;
    default false;
    description
      "The switch for the segmentation feature. When disabled, the
      publisher will not allow fragment for a very large data";
  }

  leaf max-segment-size {
    when "../enable-segmentation = 'true'";
    if-feature segmentation;
    type uint32;
    description
      "UDP-Notif provides a configurable max-segment-size to
      control the size of each segment (UDP-Notif header, with
      options, included).";
  }
}

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

```
}
<CODE ENDS>
```

8. IANA Considerations

This document describes several new registries, the URIs from IETF XML Registry and the registration of a two new YANG module names.

8.1. IANA registries

This document is creating 3 registries called "UDP-notif media types", "UDP-notif option types", and "UDP-notif header version" under the new group "UDP-notif protocol". The registration procedure is made using the Standards Action process defined in [RFC8126].

The first requested registry is the following:

```
Registry Name: UDP-notif media types
Registry Category: UDP-notif protocol.
Registration Procedure: Standard Action as defined in RFC8126
Maximum value: 15
```

These are the initial registrations for "UDP-notif media types":

```
Value: 0
Description: Reserved
Reference: RFC-to-be

Value: 1
Description: media type application/yang-data+json
Reference: <xref target="RFC8040"/>

Value: 2
Description: media type application/yang-data+xml
Reference: <xref target="RFC8040"/>

Value: 3
Description: media type application/yang-data+cbor
Reference: <xref target="RFC9254"/>
```

The second requested registry is the following:

```
Registry Name: UDP-notif option types
Registry Category: UDP-notif protocol.
Registration Procedure: Standard Action as defined in RFC8126
Maximum value: 255
```

These are the initial registrations for "UDP-notif options types":

```
  Value: 0
  Description: Reserved
  Reference: RFC-to-be

  Value: TBD1 (suggested value: 1)
  Description: Segmentation Option
  Reference: RFC-to-be

  Value: TBD2 (suggested value: 2)
  Description: Private Encoding Option
  Reference: RFC-to-be
```

The third requested registry is the following:

```
  Registry Name: UDP-notif header version
  Registry Category: UDP-notif protocol.
  Registration Procedure: Standard Action as defined in RFC8126
  Maximum value: 7
```

These are the initial registrations for "UDP-notif header version":

```
  Value: 0
  Description: UDP based Publication Channel for Streaming Telemetry
  Reference: draft-ietf-netconf-udp-pub-channel-05

  Value: 1
  Description: UDP-based Transport for Configured Subscriptions.
  Reference: RFC-to-be
```

8.2. URI

IANA is also requested to assign a two new URI from the IETF XML Registry [RFC3688]. The following two URIs are suggested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-udp-client
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

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

8.3. YANG module name

This document also requests a two new YANG module names in the YANG Module Names registry [RFC8342] with the following suggestions:

```
name: ietf-udp-client
namespace: urn:ietf:params:xml:ns:yang:ietf-udp-client
prefix: udpc
reference: RFC-to-be

name: ietf-udp-notif
namespace: urn:ietf:params:xml:ns:yang:ietf-udp-notif-transport
prefix: unt
reference: RFC-to-be
```

9. Implementation Status

Note to the RFC-Editor: Please remove this section before publishing.

9.1. Open Source Publisher

INSA Lyon implemented this document for a YANG Push publisher in an example implementation.

The open source code can be obtained here: [INSA-Lyon-Publisher].

9.2. Open Source Receiver Library

INSA Lyon implemented this document for a YANG Push receiver as a library.

The open source code can be obtained here: [INSA-Lyon-Receiver].

9.3. Pmacct Data Collection

The open source YANG push receiver library has been integrated into the Pmacct open source Network Telemetry data collection.

9.4. Huawei VRP

Huawei implemented this document for a YANG Push publisher in their VRP platform.

10. Security Considerations

[RFC8085] states that "UDP applications that need to protect their communications against eavesdropping, tampering, or message forgery SHOULD employ end-to-end security services provided by other IETF protocols". As mentioned above, the proposed mechanism is designed to be used in controlled environments, as defined in [RFC8085] also known as "limited domains", as defined in [RFC8799]. Thus, a security layer is not necessary required. Nevertheless, a DTLS layer MUST be implemented in open or unsecured networks. A specification

of udp-notif using DTLS is presented in Section 6.

11. Acknowledgements

The authors of this documents would like to thank Alexander Clemm, Benoit Claise, Eric Voit, Huiyang Yang, Kent Watsen, Mahesh Jethanandani, Marco Tollini, Hannes Tschofenig, Rob Wilton, Sean Turner, Stephane Frenot, Timothy Carey, Tim Jenkins, Tom Petch and Yunan Gu for their constructive suggestions for improving this document.

12. References

12.1. Normative References

[I-D.ietf-netconf-distributed-notif]
Zhou, T., Zheng, G., Voit, E., Graf, T., and P. Francois, "Subscription to Distributed Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-distributed-notif-08, 6 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-distributed-notif-08>.

[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-13, 4 November 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-13>.

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tls-client-server-29, 18 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-29>.

[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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

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

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

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

[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>.

[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>.

[RFC8650] Voit, E., Rahman, R., Nilsen-Nygaard, E., Clemm, A., and A. Bierman, "Dynamic Subscription to YANG Events and Datastores over RESTCONF", RFC 8650, DOI 10.17487/RFC8650, November 2019, <https://www.rfc-editor.org/info/rfc8650>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

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

12.2. Informative References

[I-D.ahuang-netconf-notif-yang]
Feng, A. H., Francois, P., Graf, T., and B. Claise, "YANG model for NETCONF Event Notifications", Work in Progress, Internet-Draft, draft-ahuang-netconf-notif-yang-02, 23 July 2023, <https://datatracker.ietf.org/doc/html/draft-ahuang-netconf-notif-yang-02>.

[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://datatracker.ietf.org/doc/html/draft-ietf-netconf-notification-messages-08>.

[INSA-Lyon-Publisher]
"INSA Lyon, YANG Push publisher example implementation", <https://github.com/network-analytics/udp-notif-scapy>.

[INSA-Lyon-Receiver]
"INSA Lyon, YANG Push receiver library implementation", <https://github.com/network-analytics/udp-notif-c-collector>.

[Paolo-Lucente-Pmacct]
"Paolo Lucente, Pmacct open source Network Telemetry Data Collection", <https://github.com/pmacct/pmacct>.

[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>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[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>.

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

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

[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>.

[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>.

[RFC8799] Carpenter, B. and B. Liu, "Limited Domains and Internet Protocols", RFC 8799, DOI 10.17487/RFC8799, July 2020, <https://www.rfc-editor.org/info/rfc8799>.

Appendix A. UDP-notif Examples

This non-normative section shows two examples of how the the "ietf-udp-notif-transport" YANG module can be used to configure a [RFC8639] based publisher to send notifications to a receiver and an example of a YANG Push notification message using UDP-notif transport protocol.

A.1. Configuration for UDP-notif transport with DTLS disabled

This example shows how UDP-notif can be configured without DTLS encryption.

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

   <?xml version='1.0' encoding='UTF-8'?>
   <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <subscriptions xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-\
   notifications">
       <subscription>
         <id>6666</id>
         <stream-subtree-filter>some-subtree-filter</stream-subtree-fil\
   ter>
         <stream>some-stream</stream>
         <transport xmlns:unt="urn:ietf:params:xml:ns:yang:ietf-udp-not\
   if-transport">unt:udp-notif</transport>
         <encoding>encode-json</encoding>
         <receivers>
           <receiver>
             <name>subscription-specific-receiver-def</name>
             <receiver-instance-ref xmlns="urn:ietf:params:xml:ns:yang:\
   ietf-subscribed-notif-receivers">global-udp-notif-receiver-def</rece\
   iver-instance-ref>
           </receiver>
         </receivers>
         <periodic xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
           <period>6000</period>
         </periodic>
       </subscription>
       <receiver-instances xmlns="urn:ietf:params:xml:ns:yang:ietf-subs\
   cribed-notif-receivers">
         <receiver-instance>
           <name>global-udp-notif-receiver-def</name>
           <udp-notif-receiver xmlns="urn:ietf:params:xml:ns:yang:ietf-\
   udp-notif-transport">
             <remote-address>192.0.5.1</remote-address>
             <remote-port>12345</remote-port>
             <enable-segmentation>false</enable-segmentation>
             <max-segment-size/>
           </udp-notif-receiver>
         </receiver-instance>
       </receiver-instances>
     </subscriptions>
   </config>
```

A.2. Configuration for UDP-notif transport with DTLS enabled

This example shows how UDP-notif can be configured with DTLS encryption.

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

   <?xml version='1.0' encoding='UTF-8'?>
   <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <subscriptions xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-\
   notifications">
       <subscription>
         <id>6666</id>
         <stream-subtree-filter>some-subtree-filter</stream-subtree-fil\
   ter>
         <stream>some-stream</stream>
         <transport xmlns:unt="urn:ietf:params:xml:ns:yang:ietf-udp-not\
   if-transport">unt:udp-notif</transport>
         <encoding>encode-json</encoding>
         <receivers>
           <receiver>
             <name>subscription-specific-receiver-def</name>
             <receiver-instance-ref xmlns="urn:ietf:params:xml:ns:yang:\
   ietf-subscribed-notif-receivers">global-udp-notif-receiver-dtls-def<\
   /receiver-instance-ref>
           </receiver>
         </receivers>
         <periodic xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
           <period>6000</period>
         </periodic>
       </subscription>
       <receiver-instances xmlns="urn:ietf:params:xml:ns:yang:ietf-subs\
   cribed-notif-receivers">
         <receiver-instance>
           <name>global-udp-notif-receiver-dtls-def</name>
           <udp-notif-receiver xmlns="urn:ietf:params:xml:ns:yang:ietf-\
   udp-notif-transport">
             <remote-address>192.0.5.1</remote-address>
             <remote-port>12345</remote-port>
             <enable-segmentation>false</enable-segmentation>
             <max-segment-size/>
             <dtls>
               <client-identity>
                 <tls13-epsk>
                   <local-definition>
                     <key-format>ct:octet-string-key-format</key-format>
                     <cleartext-key>BASE64VALUE=</cleartext-key>
                   </local-definition>
                   <external-identity>example_external_id</external-ide\
   ntity>
                   <hash>sha-256</hash>
                   <context>example_context_string</context>
                   <target-protocol>8443</target-protocol>
```

```
                  <target-kdf>12345</target-kdf>
                </tls13-epsk>
              </client-identity>
              <server-authentication>
                <ca-certs>
                  <local-definition>
                    <certificate>
                      <name>Server Cert Issuer #1</name>
                      <cert-data>BASE64VALUE=</cert-data>
                    </certificate>
                    <certificate>
                      <name>Server Cert Issuer #2</name>
                      <cert-data>BASE64VALUE=</cert-data>
                    </certificate>
                  </local-definition>
                </ca-certs>
                <ee-certs>
                  <local-definition>
                    <certificate>
                      <name>My Application #1</name>
                      <cert-data>BASE64VALUE=</cert-data>
                    </certificate>
                    <certificate>
                      <name>My Application #2</name>
                      <cert-data>BASE64VALUE=</cert-data>
                    </certificate>
                  </local-definition>
                </ee-certs>
                <raw-public-keys>
                  <local-definition>
                    <public-key>
                      <name>corp-fw1</name>
                      <public-key-format>ct:subject-public-key-info-fo\
rmat</public-key-format>
                      <public-key>BASE64VALUE=</public-key>
                    </public-key>
                    <public-key>
                      <name>corp-fw2</name>
                      <public-key-format>ct:subject-public-key-info-fo\
rmat</public-key-format>
                      <public-key>BASE64VALUE=</public-key>
                    </public-key>
                  </local-definition>
                </raw-public-keys>
                <tls13-epsks/>
              </server-authentication>
              <keepalives>
                <test-peer-aliveness>
```

```
                  <max-wait>30</max-wait>
                  <max-attempts>3</max-attempts>
                </test-peer-aliveness>
              </keepalives>
            </dtls>
          </udp-notif-receiver>
        </receiver-instance>
      </receiver-instances>
    </subscriptions>
  </config>
```

A.3. YANG Push message with UDP-notif transport protocol

This example shows how UDP-notif is used as a transport protocol to send a "push-update" notification [RFC8641] encoded in JSON [RFC7951].

Assuming the publisher needs to send the JSON payload showed in Figure 6, the UDP-notif transport is encoded following the Figure 7. The UDP-notif message is then encapsulated in a UDP frame.

```
{
    "ietf-notification:notification": {
        "eventTime": "2023-02-10T08:00:11.22Z",
        "ietf-yang-push:push-update": {
            "id": 1011,
            "datastore-contents": {
                "ietf-interfaces:interfaces": [
                    {
                        "interface": {
                            "name": "eth0",
                            "oper-status": "up"
                        }
                    }
                ]
            }
        }
    }
}
```

Figure 6: JSON Payload to be sent

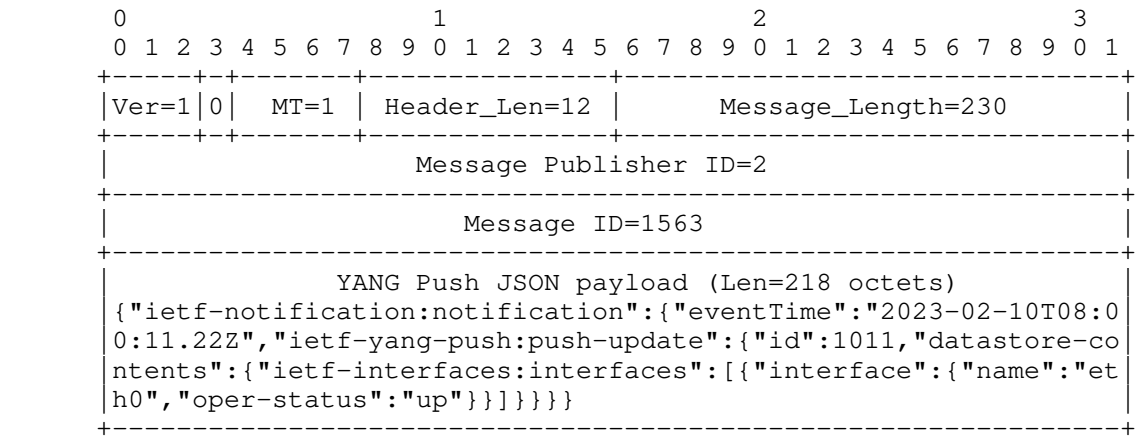


Figure 7: UDP-notif transport message

Authors' Addresses

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

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

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

Paolo Lucente
NTT
Siriusdreef 70-72
Hoofddorp, WT 2132
Netherlands
Email: paolo@ntt.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 20 April 2024

K. Larsson
Deutsche Telekom

18 October 2023

Mapping YANG Data to Label-Set Time Series
draft-kll-yang-label-tsdb-00

Abstract

This document proposes a standardized approach for representing YANG-modeled configuration and state data, for storage in Time Series Databases (TSDBs) that identify time series using a label-set. It outlines procedures for translating YANG data representations to fit within the label-centric structures of TSDBs and vice versa. This mapping ensures clear and efficient storage and querying of YANG-modeled data in TSDBs.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/plajjan/draft-kll-yang-label-tsdb.

Status of This Memo

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

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

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

This Internet-Draft will expire on 20 April 2024.

Table of Contents

1. Introduction

The aim of this document is to define rules for representing configuration and state data defined using the YANG data modeling language [RFC7950] as time series using a label-centric model.

The majority of modern Time Series Databases (TSDBs) employ a label-centric model. In this structure, time series are identified by a set of labels, each consisting of a key-value pair. These labels facilitate efficient querying, aggregation, and filtering of data over time intervals. Such a model contrasts with the hierarchical nature of YANG-modeled data. The challenge, therefore, lies in ensuring that YANG-defined data, with its inherent structure and depth, can be seamlessly integrated into the flat, label-based structure of most contemporary TSDBs.

This document seeks to bridge this structural gap, laying out rules and guidelines to ensure that YANG-modeled configuration and state data can be effectively stored, queried, and analyzed within label-centric TSDBs.

2. Specification of the Mapping Procedure

Instances of YANG data nodes are mapped to metrics. Only nodes that carry a value are mapped. This includes leafs and presence containers. The hierarchical path to a value, including non-presence containers and lists, form the path that is used as the name of the metric. The path is formed by joining YANG data nodes using _. Special symbols, e.g. -, in node names are replaced with _.

List keys are mapped into labels. The path to the list key is transformed in the same way as the primary name of the metric. Compound keys have each key part as a separate label.

2.1. Example: Packet Counters in IETF Interfaces Model

Consider the in-unicast-pkts leaf from the IETF interfaces model that captures the number of incoming unicast packets on an interface:

Original YANG Instance-Identifier: yang /interfaces/interface[name='eth0']/statistics/in-unicast-pkts

Following the mapping rules defined:

1. The path components, including containers and list names, are transformed into the metric name by joining the node names with _. Special symbols, e.g. - are replaced with _.

Resulting Metric Name: interfaces_interface_statistics_in_unicast_pkts

1. The list key "predicate", which in this case is the interface name (eth0), is extracted and stored as a separate label. The label key represents the complete path to the key.

Resulting Label: interfaces_interface_name = eth0

1. The leaf value, which represents the actual packet counter, remains unchanged and is directly mapped to the value in the time series database.

For instance, if the packet counter reads 5,432,100 packets:

Value: 5432100

1. As part of the standard labels, a server identification string is also included. A typical choice of identifier might be the hostname. For this example, let's assume the device name is router-01:

Label: host = router-01

Final Mapping in the TSDB:

* Metric: interfaces_interface_statistics_in_unicast_pkts
* Value: 5432100
* Labels:
  - host = router-01
  - interfaces_interface_name = eth0

2.2. Mapping values

Leaf values are mapped based on their intrinsic type:

* All integer types are mapped to integers and retain their native representation
  - some implementations only support floats for numeric values
* decimal64 values are mapped to floats and the value should be rounded and truncated as to minimize the loss of information
* Enumeration types are mapped using their string representation.
* String types remain unchanged.

2.3. Choice

Choice constructs from YANG are disregarded and not enforced during the mapping process. Given the temporal nature of TSDBs, where data spans across time, different choice branches could be active in a single data set, rendering validation and storage restrictions impractical.

2.4. Host / device name

There is an implicit host label identifying the server, typically set to the name of the host originating the time series data.

Instance data retrieved from YANG-based servers do not generally identify the server it originates from. As a time series database is likely going to contain data from multiple servers, the host label is used to identify the source of the data.

3. Querying YANG modeled time series data

The process of storing YANG-modeled data in label-centric TSDBs, as defined in the previous sections, inherently structures the data in a way that leverages the querying capabilities of modern TSDBs. This chapter provides guidelines on how to construct queries to retrieve this data effectively.

3.1. 1. *Basic Queries*

To retrieve all data points related to incoming unicast packets from the IETF interfaces model:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts

3.2. 2. *Filtering by Labels*

To retrieve incoming unicast packets specifically for the interface eth0:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE interfaces_interface_name = 'eth0'

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts{interfaces_interface_name="eth0"}

Similarly, to filter by device / host name:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE host = 'router-01'

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts{host="router-01"}

3.3. 3. *Time-based Queries*

* *InfluxQL*: sql SELECT * FROM interfaces_interface_statistics_in_unicast_pkts WHERE time > now() - 24h

Prometheus fetches data based on the configured scrape interval and retention policies, so time-based filters in PromQL often center around the range vectors. For data over the last 24 hours:

* *PromQL*: promql interfaces_interface_statistics_in_unicast_pkts[24h]

3.4. 4. *Aggregations*

To get the average number of incoming unicast packets over the last hour:

* *InfluxQL*: sql SELECT MEAN(value) FROM interfaces_interface_statistics_in_unicast_pkts WHERE time > now() - 1h GROUP BY time(10m)

* *PromQL*: promql avg_over_time(interfaces_interface_statistics_in_unicast_pkts[1h])

3.5. 5. *Combining Filters*

To retrieve the sum of incoming unicast packets for eth0 on router-01 over the last day:

* *InfluxQL*: sql SELECT SUM(value) FROM interfaces_interface_statistics_in_unicast_pkts WHERE interfaces_interface_name = 'eth0' AND host = 'router-01' AND time > now() - 24h

* *PromQL*: promql sum(interfaces_interface_statistics_in_unicast_pkts{interfaces_interface_name="eth0", host="router-01"})[24h]

3.6. 6. *Querying Enumeration Types*

In YANG models, enumerations are defined types with a set of named values. The oper-status leaf in the IETF interfaces model is an example of such an enumeration, representing the operational status of an interface.

For instance, the oper-status might have values such as up, down, or testing.

To query interfaces that have an oper-status of up:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_oper_status WHERE value = 'up'

* *PromQL*: promql interfaces_interface_oper_status{value="up"}

Similarly, to filter interfaces with oper-status of down:

* *InfluxQL*: sql SELECT * FROM interfaces_interface_oper_status WHERE value = 'down'

* *PromQL*: promql interfaces_interface_oper_status{value="down"}

This approach allows us to effectively query interfaces based on their operational status, leveraging the enumeration mapping within the TSDB.

4. Requirements on time series databases

This document specifies a mapping to a conceptual representation, not a particular concrete interface. To effectively support the mapping of YANG-modeled data into a label-centric model, certain requirements must be met by the Time Series Databases (TSDBs). These requirements ensure that the data is stored and retrieved in a consistent and efficient manner.

4.1. Support for String Values

Several YANG leaf types carry string values, including the string type itself and all its descendants as well as enumerations which are saved using their string representation.

The chosen TSDB must support the storage and querying of string values. Not all TSDBs inherently offer this capability, and thus, it's imperative to ensure compatibility.

4.2. Sufficient Path Length

YANG data nodes, especially when representing deep hierarchical structures, can result in long paths. When transformed into metric names or labels within the TSDB, these paths might exceed typical character limits imposed by some databases. It's essential for the TSDB to accommodate these potentially long names to ensure data fidelity and avoid truncation or loss of information.

4.3. High Cardinality

Given the possibility of numerous unique label combinations (especially with dynamic values like interface names, device names, etc.), the chosen TSDB should handle high cardinality efficiently. High cardinality can impact database performance and query times, so it's essential for the TSDB to have mechanisms to manage this efficiently.

5. Normative References

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

Author's Address

Kristian Larsson
Deutsche Telekom
Email: kristian@spritelink.net

NETMOD
Internet-Draft
Intended status: Standards Track
Expires: 22 April 2024

J. Lindblad
Cisco

20 October 2023

Philatelist, YANG-based collection and aggregation framework integrating Telemetry data and Time Series Databases
draft-lindblad-tlm-philatelist-00

Abstract

Timestamped telemetry data is collected en masse today. Mature tools are typically used, but the data is often collected in an ad hoc manner. While the dashboard graphs look great, the resulting data is often of questionable quality, not well defined, and hard to compare with seemingly similar data from other organizations.

This document proposes a standard, extensible, cross domain framework for collecting and aggregating timestamped telemetry data in a way that combines YANG, metadata and Time Series Databases to produce more dependable and comparable results.

About This Document

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

The latest revision of this draft can be found at https://janlindblad.github.io/netmod-tlm-philatelist/draft-lindblad-tlm-philatelist.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-lindblad-tlm-philatelist/.

Source for this draft and an issue tracker can be found at https://github.com/janlindblad/netmod-tlm-philatelist.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

1.1. The Problem

Many organizations today are collecting large amounts of telemetry data from their networks and data centers for a variety of purposes. Much (most?) of this data is funneled into a Time Series Database (TSDB) for display in a dashboard or further (AI-backed) processing and decision making.

While this data collection is often handled using standard tools, there generally seems to be little commonality when it comes to what is meaured, how the data is aggregated, or definitions of the measured quantities (if any).

Data science issues like adding overlapping quantities, adding quantities of different units of measurement, or quantities with different scopes, are likely common. Such errors are hard to detect given the ad hoc nature of the collection. This often leads to uncertainty regarding the quality of the conclusions drawn from the collected data.

1.2. The Solution

The Philatelist framework proposes to standardize the collection, definitions of the quantities measured and meta data handling to provide a robust ground layer for telemetry collection. The architecture defines a few interfaces, but allows great freedom in the implementations with its plug-in architecture. This allows flexibility enough that any kind of quantitiy can be measured, any kind of collection protocol and mechanism employed, and the data flows aggregated using any kind of operation.

To do this, YANG is used both to describe the quantities being measured, as well as act as the framework for the metadata management. Note that the usa of YANG here does not limit the architecture to traditional YANG-based transport protocols. YANG is used to describe the data, regardless of which format it arrives in.

Initially developed in context of the Power and Energy Efficiency work (POWEFF), we realized both the potential and the need for this collection and aggregation architecture to become a general framework for collection of a variety of metrics.

There is not much point in knowing the "cost side" of a running system (as in energy consumption or CO2-emissions) if that cannot be weighed against the "value side" delivered by the system (as in transported bytes, VPN connections, music streaming hours, or number of cat videos, etc.), which means traditional performance metrics will play an equally important role in the collection.

In this initial version, we have done nothing to pull the proposed YANG modules out of its POWEFF roots and generalize it for general telemetry. We believe the ideas and merits of this framework architecture will be apparent nonetheless in this first version. For the next version, we certainly need to generalize the quantities measured and rename the YANG modules and node names.

1.3. The Philatelist Name

This specification is about a framework for collection, aggregation and interpretation of timestamped telemetry data. The definition of "philatelist" seems close enough.

1. philatelist

noun. [’flætlst’] a collector and student of postage stamps.

Synonyms
- collector
- aggregator

Figure 1: Source: https://www.synonym.com/synonyms/philatelist

2. Conventions and Definitions

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

This document uses the terminology defined in [RFC7950].

In addition, this document defines the following terms:

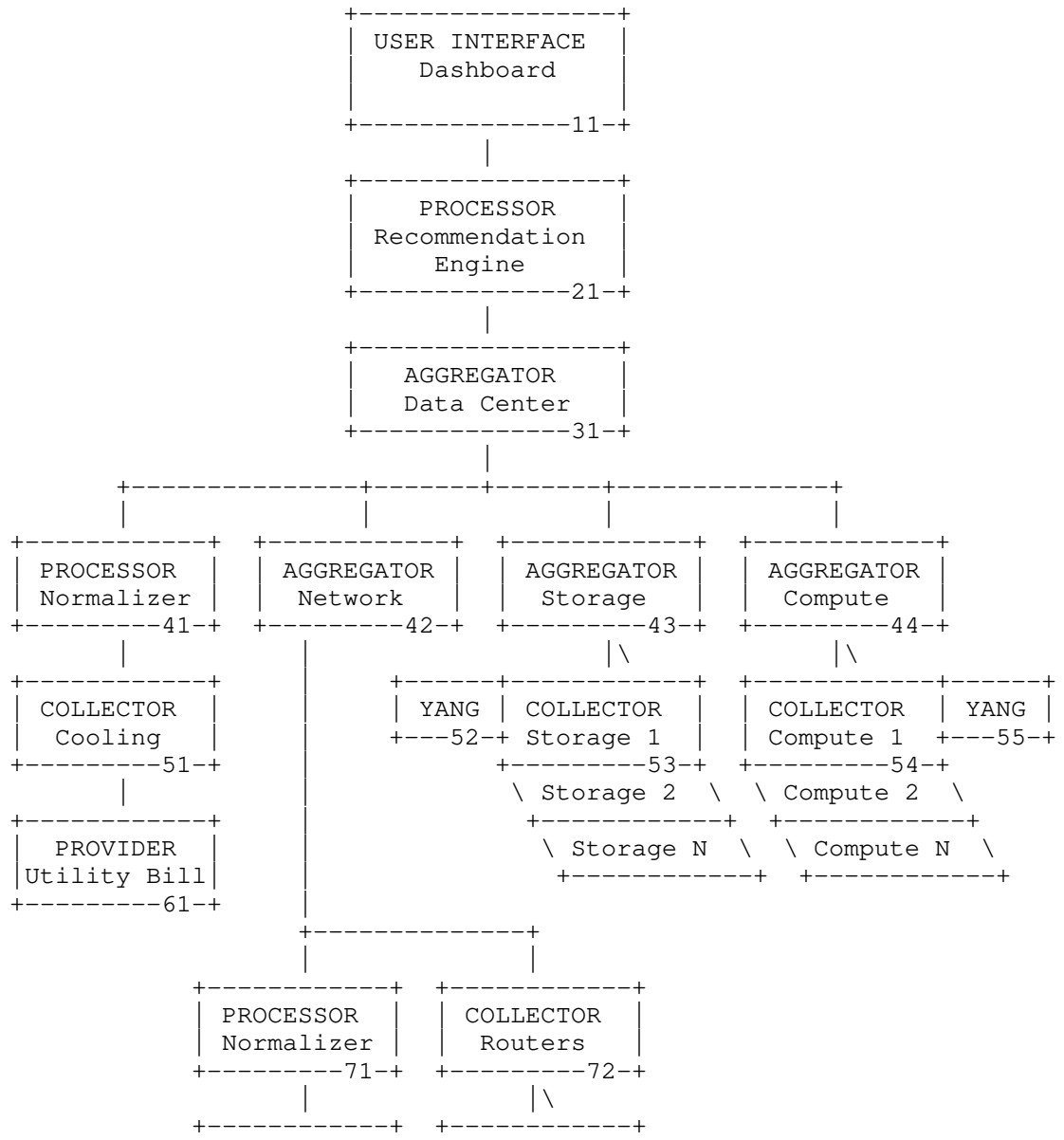
TSDB Time Series Database.

Sensor An entity in a system that delivers a snapshot value of some quantity pertaining to the system. Sensors are identified by their Sensor Path.

Sensor Path A textual representation of the sensor’s address within the system.

3. Architecure Overview

The deployment of a Philatelist framework consists of a collection of plug-in compomnents with well defined interfaces. Here is an example of a deployment. Each box is numbered in the lower right for easy reference.



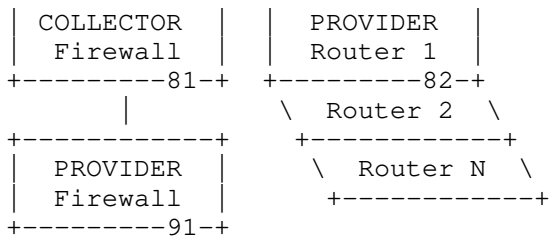


Figure 2: Example Philatelist component deployment.

Each component in the above diagram, represents a logical function. Many boxes could be running within a single server, or they could be fully distrubuted, or anything in between.

Provider components (61, 82, 91) are running on a telemetry source system that supports a YANG-based telemetry data server. The telemetry data flows from the telemetry source system to a Time Series Database (TSDB).

Collector components (51, 72, 81) ensure the Providers are programmed properly to deliver the telemetry data to the TSDB designated by the collector. In some cases this flow may be direct from the source to the TSDB, in other cases, it may be going through the collector. In some cases the collector may be polling the source, in others it may have set up an automatic, periodic subscription.

Many telemetry source systems will not have any on-board YANG-based telemetry server. Such servers will instead be managed by a collector specialized to handle a particular kind of source server (53, 54). These specialized collectors are still responsible to set up a telemetry data stream from them to the collector's TSDB. In this case, the collector will also supply a YANG description (52, 55) of the incoming data stream.

Processor components (21, 41, 71) are transforming the data stream in some way, e.g. converting from one unit of measurement to another, or adjusting the data values recorded to also include some aspect that this particular sensor is not taking into account.

Aggregator components (31, 42, 43, 44) combine the time series telemetry data flows using some operation, e.g. summing, averaging or computing the max or min over them. In this example there are aggregators for Network, Storage, Compute and the entire Data Center

On top of the stack, we may often find a (graphical) user interface (11), for human consumption of the intelligence acquired by the system. Equally relevant is of course an (AI) application making decisions based on findings in the aggregated telemetry flow.

3.1. The Provider Component

A Provider is a source of telemetry data that also offers a YANG-based management interface. Each provider typically has a large number of "sensors" that can be polled or in some cases subscribed to.

One problem with the sensors is that they are spread around inside the source system, and may not be trivial to locate. Also, the metadata asscociated with the sensor is often only missing or only available in human readable form (free form strings), rather than in a strict machine parsable format.

```
/hardware/component[name="psu3"]/.../sensor-data/value
...
/interfaces/interface[name="eth0/0"]/.../out-broadcast-packets
...
/routing/mpls/mpls-label-blocks/.../inuse-labels-count
...
```

Figure 3: Example of scattered potential sensors in a device.

To solve these problems, the Provider YANG module contains a sensor-catalog list. Essentially a list of all interesting sensors available on the system, with their sensor paths and machine parsable units, definition and any other metadata.

An admin user or application can then copy the sensor definition from the sensor catalog and insert into the configuration in the colletor.

```
  +--ro sensor-catalog
      +--ro sensors
        +--ro sensor* [path]
            +--ro path?                    xpath
            +--ro sensor-type?             identityref
            +--ro sensor-location?         something
            +--ro sensor-state?            something
            +--ro sensor-current-reading?  something
            +--ro sensor-precision?        string
```

Figure 4: YANG tree diagram of the Provider sensor-catalog.

Note: The "something" YANG-type is used in many places in this document. That is just a temporarty placeholder we use until we have figured out what the appropriate type should be.

The sensor types are defined as YANG identities, making them maximally extensible. Examples of sensor types might be energy measured in kWh, or energy measured in J, or temperature measured in F, or in C, or in K.

3.2. The Collector Component

Collector components collect data points from sources, typically by periodic polling or subscriptions, and ensure the collected data is stored in a Time Series Database (TSDB). The actual data stream may or may not be passing through the collector component; the collector is responsible for ensuring data flows from the source to the destination TSDB and that the data has a YANG description and is tagged with necessary metadata. How the collector agrees with a source to deliver data in a timely manner is beyond the scope of this document.

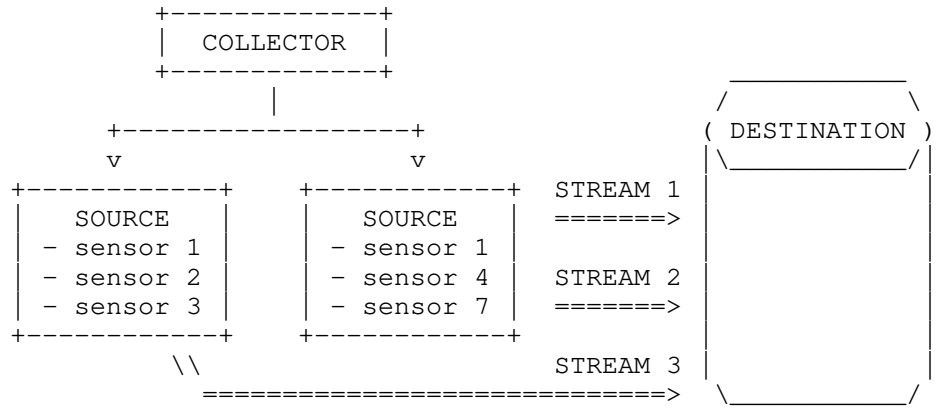


Figure 5: Example of Collector setting up three streams of telemetry data from two sources to one desitination.

Each source holds a number of sensors that may be queried or subscribed to. The collector arranges the sensors into sensour groups that presumably are logically related, and that are collected using the same method. A number of collection methods (some YANG-based, some not) are modeled directly in the ietf-poweff-collector.yang module, but the set is designed to be easily extensible.

```
+-- sensor-groups
|  +-- sensor-group* [id]
|     +-- id?                                  something
|     +-- method?                              identityref
|     +-- get-static-url-once
|     |  +-- url?                              something
|     |  +-- format?                           something
|     +-- gnmi-polling
|     |  +-- encoding?                         something
|     |  +-- protocol?                         something
|     +-- restconf-get-polling
|     |  +-- xxx?                              something
|     +-- netconf-get-polling
|     |  +-- xxx?                              something
|     +-- restconf-yang-push-subscription
|     |  +-- xxx?                              something
|     +-- netconf-yang-push-subscription
|     |  +-- xxx?                              something
|     +-- redfish-polling
|     |  +-- xxx?                              something
|     +-- frequency?                           sample-frequency
|     +-- path* [path]
|        +-- path?                             xpath
|        +-- sensor-type?                      identityref
+-- streams
  +-- stream* [id]
      +-- id?                                  something
      +-- source*                              string
      +-- sensor-group* [name]
      |  +-- name?   -> ../../../sensor-groups/sensor-group/id
      +-- destination?    -> ../../../destinations/destination/id
```

Figure 6: YANG tree diagram of the Collector sensor-groups and streams.

The sensor groups are then arranged into streams from a collection of sources (that support the same set of sensor groups) to a destination. This structure has been chosen with the assumption that there will be many source devices with the same set of sensor groups, and we want to minimize repetition.

3.3. The Processor and Aggregator Components

Processor components take an incoming data flow and transforms it somehow, and possibly augments it with a flow of derived information. The purpose of the transformation could be to convert between different units of measurement, correct for known errors in in the input data, or fill in approximate values where there are holes in the input data.

Aggregator components take multiple incoming data flows and combine them, typically by adding them together, taking possible differences in cadence in the input data flows into account.

Processor and Aggregator components provide a YANG model of the output data, just like the Collector components, so that all data flowing in the system has a YANG description and is associated with metadata.

Note: In the current version of the YANG modules, a Processor is simply an Aggregator with a single input and output. Unless we see a need to keep these two component types separate, we might remove the Processor component and keep it baked in with the Aggregator.

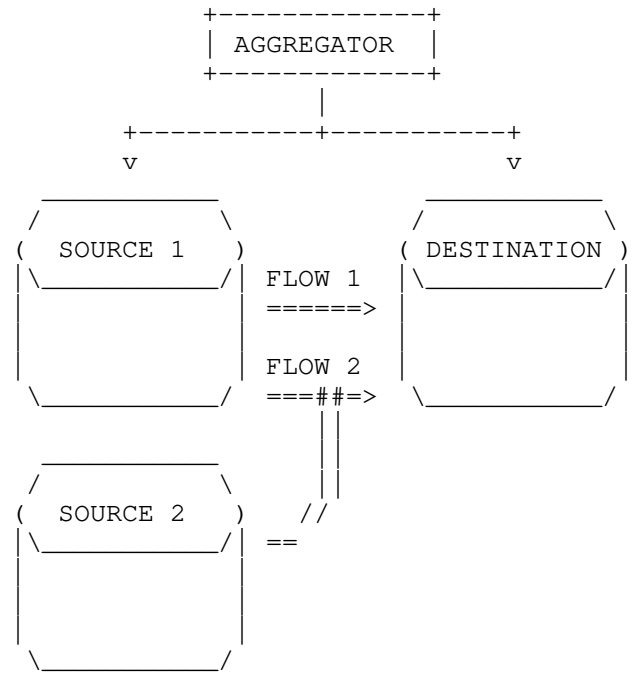


Figure 7: Example of an Aggregator setting up two flows of telemetry data from two sources to one desitination.

In this diagram, the sources and destination look like separate TSDBs, which they might be. They may also be different buckets within the same TSDB.

Each flow is associated with one or more inputs, one output and a series of processing operations. Each input flow and output flow may have an pre-processing or post-processing operation applied to it separately. Then all the input flows are combined using one or more aggregation operations. Some basic operations have been defined in the Aggregator YANG module, but the set of operations has been designed to be maximally extensible.

```
+-- flows
|  +-- flow* [id]
|     +-- id?                                    string
|     +-- (chain-position)?
|        +--:(input)
|        |  +-- input
|        |     +-- source?
|        |           -> ../../../../../sources/source/id
|        +--:(output)
|        |  +-- output
|        |     +-- destination?
|        |           -> ../../../../../destinations/destination/id
|        +--:(middle)
|           +-- middle
|              +-- inputs*
|              |     -> ../../../../flows/flow/id
|              +-- pre-process-inputs?
|              |     -> ../../../../operations/operation/id
|              +-- aggregate?
|              |     -> ../../../../operations/operation/id
|              +-- post-process-output?
|                    -> ../../../../operations/operation/id
+-- operations
  +-- operation* [id]
     +-- id?                                    something
     +-- (op-type)?
       +--:(linear-sum)
       |  +-- linear-sum
       +--:(linear-average)
       |  +-- linear-average
       +--:(linear-max)
       |  +-- linear-max
       +--:(linear-min)
       |  +-- linear-min
       +--:(rolling-average)
       |  +-- rolling-average
       |     +-- timespan?                      something
       +--:(filter-age)
       |  +-- filter-age
       |     +-- min-age?                       something
       |     +-- max-age?                       something
       +--:(function)
           +-- function
             +-- name?                          something
```

Figure 8: YANG tree diagram of the Aggregator flows and operations.

The operations listed above are basic aggregation operations. Linear-sum is just adding all the input sources together, with linear interpolation when their data points don't align perfectly in time. Rolling average is averaging the input flows over a given length of time. The filter-age drops all data points that are outside the min to max age. The function allows plugging in any other function the Aggregator may have defined, but more importantly, the operations choice is easily extended using YANG augment to include any other IETF or vendor specified extensions.

4. YANG-based Telemetry Outlook

Much work has already gone into the area of telemetry, YANG, and even their intersection. E.g. [I-D.draft-ietf-opsawg-collected-data-manifest-01] and [I-D.draft-claise-netconf-metadata-for-collection-03] come to mind.

Even though this work has a solid foundation and shares many or most of the goals with this work, we (the POWEFF team) have not found it easy to apply the above work directly in the practical work we do. So what we have tried to do is a very pragmatic approach to telemetry data collection the way we see it happening on the ground combined with the benefits of Model Driven Telemetry (MDT), in practice meaning YANG-based with additional YANG-modeled metadata.

Many essential data sources in real world deployments do not support any YANG-based interfaces, and that situation is expected to remain for the forseable future, which is why we find it important to be able to ingest data from free form (often REST-based) interfaces, and then add the necessary rigor on the Collector level. Then output the datastreams in formats that existing, mature tools can consume directly.

In particular, this draft depends on the mapping of YANG-based structures to the typical TSDB tag-based formats described in [I-D.draft-kll-yang-label-tsdb-00].

For the evolution of the YANG-based telemetry area, we believe this approach, combining pragmatism in the data flow interfaces with rigor regarding the data content, is key. We would like to make this work fit in with the works of others in the field.

5. YANG Modules

5.1. Base types module for Philatelist

```
<CODE BEGINS>
module ietf-poweff-types {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-types";
  prefix ietf-poweff-types;

  organization
    "IETF OPSA (Operations and Management Area) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>
     Editor:  Jan Lindblad
              <mailto:jlindbla@cisco.com>
     Editor:  Snezana Mitrovic
              <mailto:snmitrov@cisco.com>
     Editor:  Marisol Palmero
              <mailto:mpalmero@cisco.com>";
  description
    "This YANG module defines basic quantities, measurement units
    and sensor types for the POWEFF framework.

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

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions

     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

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

  revision 2023-10-12 {
    description
      "Initial revision of POWEFF types";
    reference
      "RFC XXXX: ...";
  }

  typedef something { // FIXME: Used when we haven't decided the type yet
    type string;
  }
  typedef xpath {
    type string; // FIXME: Proper type needed
```

```
  }
  typedef sample-frequency {
    type string; // FIXME: Proper type needed
  }

  // ========== SENSOR-CLASS ==============================
  identity sensor-class {
    description "Sensor's relation to the asset it sits on.";
  }
  identity sc-input {
    base sensor-class;
    description "Sensor reports input quantity of the asset it sits
      on.";
  }
  identity sc-output {
    base sensor-class;
    description "Sensor reports output quantity of the asset it sits
      on.";
  }
  identity sc-allocated {
    base sensor-class;
    description "Sensor reports (maximum) allocated quantity of the
      asset it sits on.";
  }

  // ========== SENSOR-QUANTITY ===========================
  identity sensor-quantity {
    description "Sensor's quantity being measured.";
  }
  identity sq-voltage {
    base sensor-quantity;
    description "Sensor reports electric tension, voltage.";
  }
  identity sq-current {
    base sensor-quantity;
    description "Sensor reports electric current.";
  }
  identity sq-power {
    base sensor-quantity;
    description "Sensor reports power draw (energy per unit of time).";
  }
  identity sq-power-apparent {
    base sq-power;
    description "Sensor reports apparent power, i.e. average electrical
      current times voltage (in VA).";
  }
  identity sq-power-true {
    base sq-power;
```

```
      description "Sensor reports true power, i.e. integral over current
        and voltage at each instant in time.";
    }
    identity sq-energy {
      base sensor-quantity;
      description "Sensor reports actual energy drawn by asset.";
    }
    identity sq-co2-emission {
      base sensor-quantity;
      description "Sensor reports CO2 (carbon dioxide) emission by
        asset.";
    }
    identity sq-co2eq-emission {
      base sensor-quantity;
      description "Sensor reports CO2 (carbon dioxide) equivalent
        emission by asset.";
    }
    identity sq-temperature {
      base sensor-quantity;
      description "Sensor reports temperature of asset.";
    }

    // ========== SENSOR-UNIT ==============================
    identity sensor-unit {
      description "Sensor's unit of reporting.";
    }
    identity su-volt {
      base sensor-unit;
      base sq-voltage;
      description "Sensor unit volt, V.";
    }
    identity su-ampere {
      base sensor-unit;
      base sq-current;
      description "Sensor unit ampere, A.";
    }
    identity su-watt {
      base sensor-unit;
      base sq-power;
      description "Sensor unit watt, W.";
    }
    identity su-voltampere {
      base sensor-unit;
      base sq-power;
      description "Sensor unit Volt*Ampere, VA.";
    }
    identity su-kw {
      base sensor-unit;
```

```
      base sq-power;
      description "Sensor unit kilowatt, kW.";
    }
    identity su-joule {
      base sensor-unit;
      base sq-energy;
      description "Sensor unit joule, J.";
    }
    identity su-wh {
      base sensor-unit;
      base sq-energy;
      description "Sensor unit watthour, Wh.";
    }
    identity su-kwh {
      base sensor-unit;
      base sq-energy;
      description "Sensor unit kliowatthour, kWh.";
    }
    identity su-kelvin {
      base sensor-unit;
      base sq-temperature;
      description "Sensor unit kelvin, K.";
    }
    identity su-celsius {
      base sensor-unit;
      base sq-temperature;
      description "Sensor unit celsius, C.";
    }
    identity su-farenheit {
      base sensor-unit;
      base sq-temperature;
      description "Sensor unit farenheit, F.";
    }
    identity su-gram {
      base sensor-unit;
      base sq-co2-emission;
      description "Sensor unit gram, g.";
    }
    identity su-kg {
      base sensor-unit;
      base sq-co2-emission;
      description "Sensor unit kliogram, kg.";
    }
    identity su-ton {
      base sensor-unit;
      base sq-co2-emission;
      description "Sensor unit ton, t.";
    }
```

```
// ========== SENSOR-TYPE ==============================
identity sensor-type {
  description "Sensor's type, i.e. combination of class, quantity and
    unit.";
}
identity st-v-in {
  base sensor-type;
  base sc-input;
  base sq-voltage;
  base su-volt;
  description "Sensor reporting Voltage In to asset.";
}
identity st-v-out {
  base sensor-type;
  base sc-output;
  base sq-voltage;
  base su-volt;
  description "Sensor reporting Voltage Out of asset.";
}
identity st-i-in {
  base sensor-type;
  base sc-input;
  base sq-current;
  base su-ampere;
  description "Sensor reporting Current In to asset.";
}
identity st-i-out {
  base sensor-type;
  base sc-output;
  base sq-current;
  base su-ampere;
  description "Sensor reporting Current Out of asset.";
}
identity st-p-in-apparent-watt {
  base sensor-type;
  base sc-input;
  base sq-power-apparent;
  base su-voltampere;
  description "Sensor reporting Power In to asset as apparent (I*U)
    power.";
}
identity st-p-out-apparent-watt {
  base sensor-type;
  base sc-output;
  base sq-power-apparent;
  base su-voltampere;
  description "Sensor reporting Power Out of asset as apparent (I*U)
    power.";
```

```
    }
    identity st-p-in-true-watt {
      base sensor-type;
      base sc-input;
      base sq-power-true;
      base su-watt;
      description "Sensor reporting Power In to asset as true power.";
    }
    identity st-p-out-true-watt {
      base sensor-type;
      base sc-output;
      base sq-power-true;
      base su-watt;
      description "Sensor reporting Power Out of asset as true power.";
    }
    identity st-p-allocated-watt {
      base sensor-type;
      base sc-allocated;
      base sq-power;
      base su-watt;
      description "Sensor reporting Allocated Power for asset.";
    }
    identity st-w-j {
      base sensor-type;
      base sq-energy;
      base su-joule;
      description "Sensor reporting energy draw of asset in J.";
    }
    identity st-w-wh {
      base sensor-type;
      base sq-energy;
      base su-wh;
      description "Sensor reporting energy draw of asset in Wh.";
    }
    identity st-w-kwh {
      base sensor-type;
      base sq-energy;
      base su-kwh;
      description "Sensor reporting energy draw of asset in kWh.";
    }
    identity st-t-k {
      base sensor-type;
      base sq-temperature;
      base su-kelvin;
      description "Sensor reporting Temperature of asset in K.";
    }
    identity st-t-c {
      base sensor-type;
```

```
      base sq-temperature;
      base su-celsius;
      description "Sensor reporting Temperature of asset in °C.";
    }
    identity st-t-f {
      base sensor-type;
      base sq-temperature;
      base su-farenheit;
      description "Sensor reporting Temperature of asset in °F.";
    }

    // ========== COLLECTION-METHOD ==============================

    identity collection-method;
    identity cm-polled {
      base collection-method;
    }
    identity cm-gnmi {
      base collection-method;
    }
    identity cm-restconf {
      base collection-method;
    }
    identity cm-netconf {
      base collection-method;
    }
    identity cm-redfish {
      base collection-method;
    }
    identity get-static-url-once {
      base collection-method;
    }
    identity gnmi-polling {
      base cm-gnmi;
      base cm-polled;
    }
    identity restconf-get-polling {
      base cm-restconf;
      base cm-polled;
    }
    identity netconf-get-polling {
      base cm-netconf;
      base cm-polled;
    }
    identity restconf-yang-push-subscription {
      base cm-restconf;
    }
    identity netconf-yang-push-subscription {
```

```
      base cm-netconf;
    }
    identity redfish-polling {
      base cm-redfish;
    }
  }
  <CODE ENDS>
```

5.2. Provider interface module for Philatelist

```
  <CODE BEGINS>
  module ietf-poweff-provider {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-provider";
    prefix ietf-poweff-provider;

    import ietf-poweff-types {
      prefix ietf-poweff-types;
    }

    organization
      "IETF OPSA (Operations and Management Area) Working Group";
    contact
      "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
       WG List:  <mailto:opsawg@ietf.org>
       Editor:  Jan Lindblad
                <mailto:jlindbla@cisco.com>
       Editor:  Snezana Mitrovic
                <mailto:snmitrov@cisco.com>
       Editor:  Marisol Palmero
                <mailto:mpalmero@cisco.com>";
    description
      "This YANG module defines the POWEFF Provider.

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

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Simplified BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions

       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

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

```
     revision 2023-10-12 {
       description
         "Initial revision of POWEFF Provider";
       reference
         "RFC XXXX: ...";
     }

     grouping provider-g {
       container sensor-catalog {
         config false;
         container sensors {
           list sensor {
             key path;
             leaf path { type ietf-poweff-types:xpath; }
             leaf sensor-type { type identityref { base ietf-poweff-types:sensor-
type; }}

             leaf sensor-location {
               type ietf-poweff-types:something;
               description
                 "Indicates the current location where the sensor is located
                   in the chassis,typically refers to slot";
             }
             leaf sensor-state { // FIXME: What does this mean?
               type ietf-poweff-types:something;
               description
                 "Current state of the sensor";
             }
             leaf sensor-current-reading { // FIXME: Do we want a copy of the val
ue here?
               type ietf-poweff-types:something;
               description
                 "Current reading of the sensor";
             }
             leaf sensor-precision {
               type string;
               description
                 "Maximum deviation to be considered. This attribute mainly
                 will apply to drawn power, which corresponds to PSU PowerIn
                 measured power or calculated power; assuming discrepancy
                 between Real Power, power collected from a power meter, and
                 power measured or calculated from the metrics provided by
                 the sensors";
             }
             container sensor-thresholds { // FIXME: Is this for generating alarm
s, or what?
               description
                 "Threshold values for the particular sensor.
                 Default values shall beprovided as part of static data
                 but when configurable need to be pulledfrom the device.
                 Ideally, the sensor should allow configuing
```

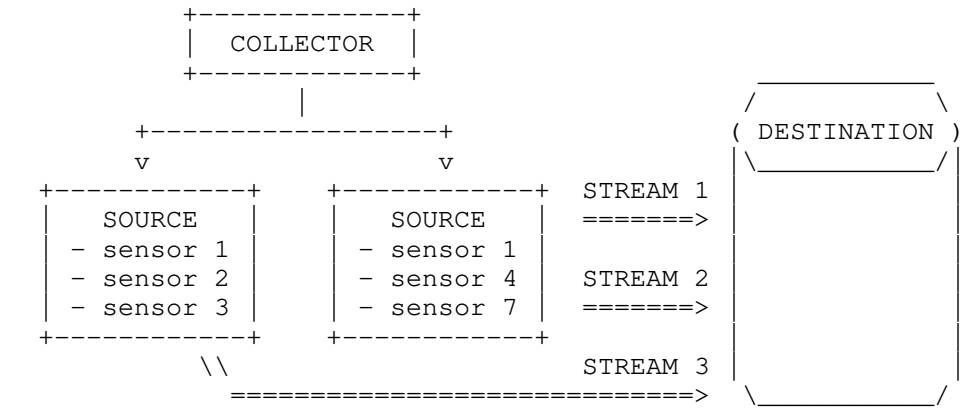
```
                thesethreshold values";

              leaf minor-low {
                type string;
                description
                  "minor-low";
              }
              leaf minor-high {
                type string;
                description
                  "minor-high";
              }
              leaf major-low {
                type string;
                description
                  "major-low";
              }
              leaf major-high {
                type string;
                description
                  "major-high";
              }
              leaf critical-low {
                type string;
                description
                  "critical-low";
              }
              leaf critical-high {
                type string;
                description
                  "critical-high";
              }
              leaf shutdown { // FIXME: What does this mean for a sensor?
                type string;
                description
                  "shutdown";
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

5.3. Collector interface module for Philatelist

```
<CODE BEGINS>
module ietf-poweff-collector {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-collector";
  prefix ietf-poweff-collector;

  import ietf-poweff-types {
    prefix ietf-poweff-types;
  }

  organization
    "IETF OPSA (Operations and Management Area) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
     WG List:  <mailto:opsawg@ietf.org>
     Editor:  Jan Lindblad
              <mailto:jlindbla@cisco.com>
     Editor:  Snezana Mitrovic
              <mailto:snmitrov@cisco.com>
     Editor:  Marisol Palmero
              <mailto:mpalmero@cisco.com>";
  description
    "This YANG module defines the POWEFF Collector.

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

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions

     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

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

  revision 2023-10-12 {
    description
      "Initial revision of POWEFF Collector";
    reference
      "RFC XXXX: ...";
  }

/*
```

A COLLECTOR programs one or more SOURCE(s) to generate a STREAM of telemetry data. The STREAM is sent to a specific DESTINATION.

Each STREAM consists of timestamped sensor values from each sensor in a sensor group.




```
*/

  grouping data-endpoint-g {
    leaf url { type ietf-poweff-types:something; }
    leaf organization { type ietf-poweff-types:something; }
    leaf bucket { type ietf-poweff-types:something; }
    container impl-specific {
      list binding {
        key key;
        leaf key { type string; }
        choice value-type {
          leaf value { type string; }
          leaf-list values { type string; ordered-by user; }
          leaf env-var { type string; }
        }
      }
    }
  }

  grouping sensor-group-g {
    leaf method {
      type identityref {
        base ietf-poweff-types:collection-method;
      }
    }
    container get-static-url-once {
```

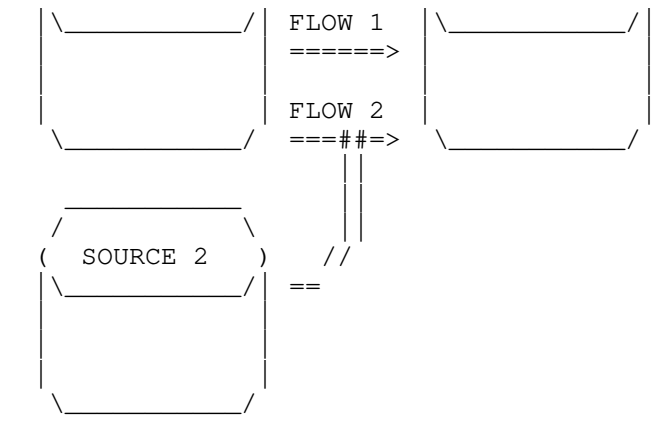
```
          when "derived-from-or-self(../method, 'ietf-poweff-types:get-static-url-
once')";
          leaf url { type ietf-poweff-types:something; }
          leaf format { type ietf-poweff-types:something; } // JSON-IETF, XML, etc
        }
        container gnmi-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:gnmi-polling')"
;
          leaf encoding { type ietf-poweff-types:something; } // self-describing-g
pb
          leaf protocol { type ietf-poweff-types:something; } // protocol grpc no-
tls
        }
        container restconf-get-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:restconf-get-po
lling')";
          leaf xxx { type string; }
        }
        container netconf-get-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:netconf-get-pol
ling')";
          leaf xxx { type string; }
        }
        container restconf-yang-push-subscription {
          when "derived-from-or-self(../method, 'ietf-poweff-types:restconf-yang-p
ush-subscription')";
          leaf xxx { type string; }
        }
        container netconf-yang-push-subscription {
          when "derived-from-or-self(../method, 'ietf-poweff-types:netconf-yang-pu
sh-subscription')";
          leaf xxx { type string; }
        }
        container redfish-polling {
          when "derived-from-or-self(../method, 'ietf-poweff-types:redfish-polling
')";
          leaf xxx { type string; }
        }
        leaf frequency {
          when "derived-from(../method, 'ietf-poweff-types:cm-polled')";
          type ietf-poweff-types:sample-frequency;
        }
        list path {
          key path;
          leaf path { type ietf-poweff-types:xpath; }
          leaf sensor-type { type identityref { base ietf-poweff-types:sensor-type
; }}
          leaf attribution { type string; }
        }
      }

      grouping collector-g {
        container poweff-collector {
          container destinations {
            list destination {
              key id;
              leaf id { type ietf-poweff-types:something; }
              uses data-endpoint-g;
```

```
          }
        }

        container sensor-groups {
          list sensor-group {
            key id;
            leaf id { type ietf-poweff-types:something; }
            uses sensor-group-g;
          }
        }

        container streams {
          list stream {
            key id;
            leaf id { type ietf-poweff-types:something; }
            leaf-list source { type string; } // Implementation specific meaning
, possibly wildcards
            list sensor-group {
              key name;
              leaf name { type leafref { path ../../../sensor-groups/sensor-grou
p/id; }}
            }
            leaf destination { type leafref { path ../../../destinations/destina
tion/id; }}
          }
        }
      }
    }
  }
  <CODE ENDS>
```

5.4. Aggregator interface module for Philatelist

```
  <CODE BEGINS>
  module ietf-poweff-aggregator {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-poweff-aggregator";
    prefix ietf-poweff-aggregator;

    import ietf-poweff-types {
      prefix ietf-poweff-types;
    }
    import ietf-poweff-collector {
      prefix ietf-poweff-collector;
    }

    organization
      "IETF OPSA (Operations and Management Area) Working Group";
    contact
      "WG Web:   <https://datatracker.ietf.org/wg/opsawg/>
       WG List:  <mailto:opsawg@ietf.org>
```

```
        Editor:  Jan Lindblad
                 <mailto:jlindbla@cisco.com>
        Editor:  Snezana Mitrovic
                 <mailto:snmitrov@cisco.com>
        Editor:  Marisol Palmero
                 <mailto:mpalmero@cisco.com>";
    description
      "This YANG module defines the POWEFF Aggregator.

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

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Simplified BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions

       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

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

    revision 2023-10-12 {
      description
        "Initial revision of POWEFF Aggregator";
      reference
        "RFC XXXX: ...";
    }

  /*

    An AGGREGATOR ensures data from one or more SOURCE(s) are
    combined into a FLOW using a (sequence of) OPERATIONs (OPs)
    to generate a new data set in the DESTINATION (which could
    be a new collection in the same data storage system as the
    SOURCE).

                  +-------------+
                  | AGGREGATOR  |
                  +-------------+
                         |
              +----------+----------+
              v                     v
       ____________           ____________
      /            \         /            \
     (  SOURCE 1    )       ( DESTINATION )
```




```
   */

     grouping aggregator-g {
       container poweff-aggregator {
         container sources {
           list source {
             key id;
             leaf id { type ietf-poweff-types:something; }
             uses ietf-poweff-collector:data-endpoint-g;
           }
         }
         container destinations {
           list destination {
             key id;
             leaf id { type ietf-poweff-types:something; }
             uses ietf-poweff-collector:data-endpoint-g;
           }
         }
         container flows {
           list flow {
             key id;
             leaf id { type string; }
             choice chain-position {
               container input {
                 leaf source { type leafref { path ../../../../../sources/source/
id; }}
               }
               container output {
                 leaf destination { type leafref { path ../../../../../destinatio
ns/destination/id; }}
               }
               container middle {
                 leaf-list inputs { type leafref { path ../../../../flows/flow/id
; }}
                 leaf pre-process-inputs { type leafref { path ../../../../operat
ions/operation/id; }}
```

```
                leaf aggregate { type leafref { path ../../../../operations/oper
ation/id; }}
                leaf post-process-output { type leafref { path ../../../../opera
tions/operation/id; }}
              }
            }
          }
        }
        container operations {
          list operation {
            key id;
            leaf id { type ietf-poweff-types:something; }
            choice op-type {
              container linear-sum {}
              container linear-average {}
              container linear-max {}
              container linear-min {}
              container rolling-average {
                leaf timespan { type ietf-poweff-types:something; }
              }
              container filter-age {
                leaf min-age { type ietf-poweff-types:something; }
                leaf max-age { type ietf-poweff-types:something; }
              }
              container function {
                leaf name { type ietf-poweff-types:something; }
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

6. Security Considerations

TODO Security

7. IANA Considerations

This document has no IANA actions.

8. References

8.1. Normative References

[I-D.draft-kll-yang-label-tsdb-00]
Larsson, K., "Mapping YANG Data to Label-Set Time Series",
Work in Progress, Internet-Draft, draft-kll-yang-label-

tsdb-00, 18 October 2023,
<https://datatracker.ietf.org/doc/html/draft-kll-yang-label-tsdb-00>.

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

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

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

8.2. Informative References

[I-D.draft-claise-netconf-metadata-for-collection-03]
Claise, B., Nayyar, M., and A. R. Sesani, "Per-Node Capabilities for Optimum Operational Data Collection", Work in Progress, Internet-Draft, draft-claise-netconf-metadata-for-collection-03, 25 January 2022, <https://datatracker.ietf.org/doc/html/draft-claise-netconf-metadata-for-collection-03>.

[I-D.draft-ietf-opsawg-collected-data-manifest-01]
Claise, B., Quilbeuf, J., Lopez, D., Martinez-Casanueva, I. D., and T. Graf, "A Data Manifest for Contextualized Telemetry Data", Work in Progress, Internet-Draft, draft-ietf-opsawg-collected-data-manifest-01, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-collected-data-manifest-01>.

Acknowledgments

Kristian Larsson has provided invaluable insights, experience and validation of the design. Many thanks to the entire POWEFF team for their committment, flexibility and hard work behind this. Hat off to Benoît Claise, who inspires by the extensive work produced in IETF over the years, and in this area in particular.

Author’s Address

Jan Lindblad
Cisco
Email: jlindbla@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 9 April 2024

T. Graf
Swisscom
J. Quilbeuf
Huawei
A. Huang Feng
INSA-Lyon

7 October 2023

Support of Hostname and Sequencing in YANG Notifications
draft-tgraf-netconf-notif-sequencing-02

Abstract

This document specifies a new YANG module that augment the NETCONF Event Notification header to support hostname, Message Publisher ID and sequence numbers to identify from which network node and at which time the message was published. This allows the collector to recognize loss, delay and reordering between the publisher and the downstream system storing the message.

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 April 2024.

Table of Contents

1. Introduction

Section 4 of [RFC5277] describes the NETCONF event notification header using a XML Schema. In the metadata of the event notification header, only the eventTime is present indicating at which time the notification message was published. For other encodings, the same schema is implemented using a YANG module in [I-D.ahuang-netconf-notif-yang]. Furthermore, in Section 3.7 of [RFC8641], the subscription ID is added to the "push-update" and "push-change-update" notification messages allowing to recognize to which xpath or sub-tree the node was subscribed to.

When the NETCONF event notification message is forwarded from the receiver to another system, such as a messaging system or a time series database where the message is stored, the transport context is lost since it is not part of the NETCONF event notification message metadata. Therefore, the downstream system is unable to associate the message to the publishing process (the exporting router), nor able to detect message loss or reordering.

Today, network operators workaround this impediment by preserving the transport source IP address and sequence numbers of the publishing process. However, this implies that this information needs to be encoded in the NETCONF event notification message which impacts the semantic readability of the message in the downstream system.

On top of that, the transport source IP address might not represent the management IP address by which the YANG push server should be known. In other terms, the source-host [RFC6470], which is the "Address of the remote host for the session" might not be the management IP address.

By extending the NETCONF Event Notification header with sysName, which could be an IP address or a DNS domain name, a reference to the YANG push publisher process and a sequence number as described in [RFC9187], the downstream system is not only able to identify from which network node, subscription, and time the message was published but also, the order of the published messages.

To correlate network data among different Network Telemetry planes as described in Section 3.1 of [RFC9232] or among different YANG push subscription types defined in Section 3.1 of [RFC8641], sysName describes from which network node the state change was observed or from when to when the data was accounted. This is essential for understanding the timely relationship among these different planes and YANG push subscription types.

2. Extend the NETCONF Event Notification Header

Besides the eventTime described in Section 2.2.1 of [RFC5277] the following metadata objects are part of a "push-update" and "push-change-update" notification message.

sysName: Describes the hostname following the ’sysName’ object definition in [RFC1213] from where the message was published from.

messagePublisherId: Message Publisher ID is a 32-bit identifier defined in [I-D.ietf-netconf-distributed-notif]. This identifier is unique to the publisher node and identifies the publishing process of the node to allow the disambiguation of an information source.

sequenceNumber: Generates a unique sequence number as described in [RFC9187] for each published message.

Figure 1 provides an example of a "push-change-update" message with the sysName, messagePublisherId and sequenceNumber. This "push-change-update" message is encoded in 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>2023-02-04T16:30:11.22Z</eventTime>
 <sysName xmlns="urn:ietf:params:xml:ns:yang:ietf-notification-sequencing">
  example-router
 </sysName>
 <messagePublisherId xmlns="urn:ietf:params:xml:ns:yang:ietf-notification-sequenc
ing">
  1
 </messagePublisherId>
 <sequenceNumber xmlns="urn:ietf:params:xml:ns:yang:ietf-notification-sequencing"
>
  187653
 </sequenceNumber>
 <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <id>1011</id>
   <datastore-contents>
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
       <interface>
         <name>eth0</name>
         <oper-status>up</oper-status>
       </interface>
     </interfaces>
   </datastore-contents>
 </push-update>
</notification>
```

Figure 1: XML Push Example for a subscription-modified notification message

3. YANG Module for Event Notifications

3.1. YANG Tree Diagram

This ietf-notification-sequencing YANG module augments the ietf-notification YANG module specified in [I-D.ahuang-netconf-notif-yang] adding the sysName and the sequenceNumber leaves as described in Section 2 of this document.

```
module: ietf-notification-sequencing

  augment-structure /inotif:notification:
    +-- sysName                 inet:host
    +-- messagePublisherId      uint32
    +-- sequenceNumber          yang:counter32
```

3.2. Full Tree View

The following is the YANG tree diagram [RFC8340] for the ietf-notification-sequencing augmentation within the ietf-notification.

```
module: ietf-notification

  structure notification:
    +-- eventTime                          yang:date-and-time
    +-- inotifseq:sysName                  inet:host
    +-- inotifseq:messagePublisherId       uint32
    +-- inotifseq:sequenceNumber           yang:counter32
```

3.3. YANG Module

```
<CODE BEGINS> file "ietf-notification-sequencing@2023-03-25.yang"
module ietf-notification-sequencing {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-notification-sequencing";
  prefix inotifseq;
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-notification {
    prefix inotif;
    reference
      "draft-ahuang-netconf-notif-yang: NETCONF Event Notification YANG";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions";
  }

  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>
                Jean Quilbeuf
                <mailto:jean.quilbeuf@huawei.com>
                Alex Huang Feng
                <mailto:alex.huang-feng@insa-lyon.fr>";

    description
      "Defines NETCONF Event Notification structure with the sysName and
      the sequenceNumber.

      Copyright (c) 2023 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

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

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

    revision 2023-03-25 {
      description
        "First revision";
      reference
        "RFC XXXX: YANG Notifications Sequencing";
    }

    sx:augment-structure "/inotif:notification" {
      leaf sysName {
        type inet:host;
        mandatory true;
        description
          "IP address or a DNS domain name from the server from which
          the message was published.";
      }
      leaf messagePublisherId {
        type uint32;
        mandatory true;
        description
          "Identifier of the publishing process generating this notification.";
      }
      leaf sequenceNumber {
        type yang:counter32;
        mandatory true;
        description
          "Unique sequence number as described in [RFC3339] for each
```

```
            published message.";
        }
      }
    }
    <CODE ENDS>
```

4. Security Considerations

The security considerations for the NETCONF Event notifications are described in [RFC5277]. This documents adds no additional security considerations.

5. IANA Considerations

5.1. IETF XML Registry

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

```
  URI: urn:ietf:params:xml:ns:yang:ietf-notification-sequencing
  Registrant Contact: The IESG.
  XML: N/A; the requested URI is an XML namespace.
```

5.2. YANG Module Name

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

```
  name: ietf-notification-sequencing
  namespace: urn:ietf:params:xml:ns:yang:ietf-notification-sequencing
  prefix: inotifseq
  reference: RFC XXXX
```

6. Operational Considerations

6.1. SysName Correlation

In order to allow data correlation among BGP Monitoring Protocol (BMP) [RFC7854] and YANG push, the same hostname value should be used as described in section 4.4 of [RFC7854] for the information TLV in the init BMP message type.

7. Acknowledgements

The authors would like to thank Rob Wilton, Nick Corran, Pierre Francois, Benoit Claise, Ahmed Elhassany and Ignacio Dominguez Martinez-Casanueva for their review and valuable comments.

8. References

8.1. Normative References

[I-D.ahuang-netconf-notif-yang]
Feng, A. H., Francois, P., Graf, T., and B. Claise, "YANG model for NETCONF Event Notifications", Work in Progress, Internet-Draft, draft-ahuang-netconf-notif-yang-02, 23 July 2023, <https://datatracker.ietf.org/doc/html/draft-ahuang-netconf-notif-yang-02>.

[I-D.ietf-netconf-distributed-notif]
Zhou, T., Zheng, G., Voit, E., Graf, T., and P. Francois, "Subscription to Distributed Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-distributed-notif-07, 7 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-distributed-notif-07>.

[RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", STD 17, RFC 1213, DOI 10.17487/RFC1213, March 1991, <https://www.rfc-editor.org/info/rfc1213>.

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

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

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

[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>.

[RFC9187] Touch, J., "Sequence Number Extension for Windowed Protocols", RFC 9187, DOI 10.17487/RFC9187, January 2022, <https://www.rfc-editor.org/info/rfc9187>.

[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>.

8.2. Informative References

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <https://www.rfc-editor.org/info/rfc6470>.

[RFC7854] Scudder, J., Ed., Fernando, R., and S. Stuart, "BGP Monitoring Protocol (BMP)", RFC 7854, DOI 10.17487/RFC7854, June 2016, <https://www.rfc-editor.org/info/rfc7854>.

[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>.

[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>.

[RFC9232] Song, H., Qin, F., Martinez-Julia, P., Ciavaglia, L., and A. Wang, "Network Telemetry Framework", RFC 9232, DOI 10.17487/RFC9232, May 2022, <https://www.rfc-editor.org/info/rfc9232>.

Authors' Addresses

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

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

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