Network Working Group
Internet-Draft
Updates: RFC5277 (if approved)
Intended status: Standards Track
Expires: 25 July 2024

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

22 January 2024

YANG model for NETCONF Event Notifications
draft-ahuang-netconf-notif-yang-04

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 25 July 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@2024-01-22.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 2024-01-22 {
       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

5.3. YANG SID-file

IANA is requested to register a new ".sid" file in the "IETF YANG SID Registry" [I-D.ietf-core-sid]:

SID range entry point: TBD
SID range size: 50
YANG module name: ietf-notification
reference: RFC-to-be

A ".sid" file is proposed in Appendix B.

6. Acknowledgements

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

7. References

7.1. Normative References

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-24, 22 December 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-sid-24>.

[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>.

[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>.

[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 examples of how XML, YANG-JSON and YANG-CBOR are encoded.

A.1. XML encoded message

This is an example of a XML-encoded notification as defined in [RFC5277].

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2022-09-02T10:59:55.32Z</eventTime>
  <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>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-encoded notification

A.2. 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 2: JSON-encoded notification

A.3. YANG-CBOR encoded message

This is an example of YANG-CBOR encoded notification. The figure Figure 3 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 3: CBOR-encoded notification using diagnostic notation

A.4. YANG-CBOR encoded message using SIDs

This is an example of YANG-CBOR encoded notification using YANG SIDs [I-D.ietf-core-sid]. The figure Figure 4 shows the message using the CBOR diagnostic notation as defined in section 3.1 of [RFC9254].

```
{
    2551: {
        1: "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 4: CBOR-encoded notification using YANG SIDs in CBOR diagnostic notation

Appendix B. .sid file

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

For CBOR encoding using YANG-SIDs identifiers, a ".sid" file is requested to IANA in Section 5.3.

```
<CODE BEGINS> file "ietf-notification@2024-01-22.sid"
{
  "ietf-sid-file:sid-file": {
    "module-name": "ietf-notification",
    "module-revision": "2024-01-22",
    "description": "NETCONF Event Notification structure",
    "dependency-revision": [
      {
        "module-name": "ietf-yang-types",
        "module-revision": "2013-07-15"
      },
      {
        "module-name": "ietf-yang-structure-ext",
        "module-revision": "2020-06-17"
      }
    ],
    "assignment-range": [
      {
        "entry-point": "2550",
        "size": "50"
      }
    ],
    "item": [
      {
        "namespace": "module",
        "identifier": "ietf-notification",
        "sid": "2550"
      },
      {
        "namespace": "data",
        "identifier": "/ietf-notification:notification",
        "sid": "2551"
      },
      {
        "namespace": "data",
        "identifier": "/ietf-notification:notification/eventTime",
        "sid": "2552"
      }
    ]
  }
}
<CODE ENDS>
```

Figure 5: .sid file for "ietf-notification" module

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
Updates: RFC5277 RFC8639 RFC7951 RFC9254 (if approved)
Intended status: Standards Track
Expires: 19 December 2024

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

17 June 2024

YANG model for NETCONF Event Notifications
draft-ahuang-netconf-notif-yang-05

Abstract

This document defines the structure of NETCONF Event Notification in a YANG model to be used in NETCONF environments. The definition of this YANG model allows the encoding of NETCONF Event Notifications in YANG compatible encodings such as JSON and 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 19 December 2024.

Table of Contents

1. Introduction

NETCONF Event Notifications [RFC5277] and YANG-Push [RFC8639] allow NETCONF [RFC6241] servers and YANG-Push publishers to send notifications to a data collection. The NETCONF client and the YANG-Push receiver decodes the message and optionally validates the header and the content before forward it to the next process. This schema validation process ensures to not break the data processing chain.

The structure of a NETCONF Event notification has been defined in [RFC5277] using a XML Schema [W3C.REC-xml-20001006] allowing NETCONF nodes to validate the header schema of the notification message when it is encoded in XML. However, when these notifications are sent using YANG-Push [RFC8639][RFC8641], they can be encoded in other encodings such as JSON [RFC7951] or CBOR [RFC9254]. In such cases, the model defined in [RFC5277] cannot be used to validate the notification header.

This document defines the content of the header of such notifications allowing implementations to validate the schema of the notifications when they are encoded in other encodings than XML. A YANG 1.1 [RFC7950] model is defined for such purposes.

This document updates [RFC5277], [RFC8639] and [RFC7951] specifying how a Notification header should be encoded. RESTCONF Notifications [RFC8040] are out of scope of this document.

2. Relationship to past documents

This section exposes the relationship to [RFC5277], [RFC8639], [RFC7951] and [RFC9254].

2.1. Relationship to RFC5277

[RFC5277] defines a mechanism for NETCONF nodes to send notifications to a collector. These are the key relationships between the current document and [RFC5277]:

* Section 2.1 of [RFC5277] defines how to configure a subscription to receive NETCONF Event Notifications. The RPCs are defined in the XML schema in Section 3.4 of [RFC5277]. This document does not update or add any new RPCs to this schema.

* The notification structure is defined in Section 4 of [RFC5277] using a XML schema. This document defines the structure of the notification in XML, JSON and CBOR in Section 4. In XML, the same structure is used.

2.2. Relationship to RFC8639

Subscribed Notifications [RFC8639] defines a mechanism on top of [RFC5277] to stream notifications from the NETCONF node. These are the key relationships between the current document and [RFC8639]:

* Section 1.4 of [RFC8639] states that the the solution uses the notification structure defined in [RFC5277]. This document replaces this structure using a YANG module allowing JSON and CBOR encodings.

2.3. Relationship to RFC7951

[RFC7951] defines how YANG data is encoded using JSON. These are the key relationship points between the current document and [RFC7951]:

* [RFC7951] does not define explicitely how a YANG notification should be encoded using JSON encoding. This document specifies the structure of such notification for JSON when these are used in a NETCONF [RFC6241] environment.

2.4. Relationship to RFC9254

[RFC9254] defines how YANG data is encoded using CBOR. These are the key relationship points between the current document and [RFC9254]:

* [RFC9254] does not define explicitely how a YANG notification should be encoded using CBOR encoding. [RFC9254] states that Notifications are container-like instances which does not follow how notifications are encoded in XML and JSON encodings. This document replaces this statement and ensures consistency between the different notification structures in YANG. This structure is to be used within NETCONF [RFC6241] environments.

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

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

[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 JSON [RFC7951] or CBOR [RFC9254].

4. NETCONF Notification structure

This section defines how NETCONF YANG Notifications are structured in XML, JSON and CBOR encodings. The same namespace "ietf-notification" is used to be compliant with [RFC5277].

4.1. XML Structure

The same structure as defined in Section 4 of [RFC5277] is used. The structure uses the XML namespace that has been defined in [RFC5277]:

urn:ietf:params:xml:ns:netconf:notification:1.0

Two child nodes within the "notification" container are expected, representing the event time and the notification payload. The "eventTime" node is defined within the same XML namespace as the "notification" element and is compliant with [RFC3339].

The name and namespace of the payload element are determined by the YANG module containing the notification statement representing the notification message.

The following example shows a "push-update" notification defined in the YANG module of YANG-Push [RFC8641] encoded in XML:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2022-09-02T10:59:55.32Z</eventTime>
  <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <id>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-encoded notification

4.2. JSON Structure

A YANG notification encoded in JSON is structured as a root "notification" container. The namespace of this container is the name of the YANG module "ietf-notification" defined in Section 5.

Two child nodes within the "ietf-notification:notification" container are expected, representing the event time and the notification payload. The "eventTime" node is defined within the same namespace as the "ietf-notification:notification" container and is compliant with [RFC3339].

The following example shows a "push-update" notification defined in the YANG module of YANG-Push [RFC8641] encoded in JSON:

```
{
    "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: JSON-encoded notification

When Notifications are implemented within RESTCONF [RFC8040] environments, the namespace of a notification stays "ietf-restconf:notification" as defined in Section 6.4 of [RFC8040].

4.3. CBOR Structure

YANG data can be represented in CBOR using Names or SIDs in keys. The following sections shows how these messages are encoded in both cases.

4.3.1. CBOR encoded messages

Notifications encoded using keys is similar to JSON encoding as defined in Section 3.3 of [RFC9254]. The key of the element can be the element itself or be namespace-qualified. In the latter case, the namespace of the notification container uses the YANG module name "ietf-notification" defined in Section 5.

Two child nodes within the "ietf-notification:notification" container are expected, representing the event time and the notification payload. The "eventTime" node is defined within the same namespace as the "ietf-notification:notification" container and is compliant with [RFC3339].

The following example shows a "push-update" notification defined in the YANG module of YANG-Push [RFC8641] encoded in CBOR using names as keys. The example uses 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 3: CBOR-encoded notification using diagnostic notation

4.3.2. CBOR encoded messages using YANG-SIDs

A Notification encoded using YANG-SIDs replaces the names of the keys of the CBOR encoded message for a 63 bit unsigned integer. This is defined in Section 3.2 of [RFC9254] and a process for SID allocation is defined in [I-D.ietf-core-sid].

Two child nodes within the root container are expected, representing the event time and the notification payload. The root container and the "eventTime" node uses a SID and the content of the "eventTime" is compliant with [RFC3339].

This is an example of YANG-CBOR encoded notification using YANG SIDs [RFC9254]. The Figure 4 shows the message using the CBOR diagnostic notation as defined in section 3.1 of [RFC9254]:

```
{
    2551: {
        1: "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 4: CBOR-encoded notification using YANG SIDs in CBOR diagnostic notation

5. YANG Module

5.1. 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
```

5.2. 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@2024-06-17.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 and RFC7950. 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.

      This module can be used to validate XML encoded notifications
      [RFC7950], JSON encoded messages [RFC7951] and CBOR encoded
      messages [RFC9254]. Refer to Section 4 of RFC XXXX for more
      details.

      Copyright (c) 2024 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

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

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

```
    revision 2024-06-17 {
      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>
```

6. Security Considerations

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

7. IANA Considerations

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

7.1. 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
```

7.2. 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
```

7.3. YANG SID-file

IANA is requested to register a new ".sid" file in the "IETF YANG SID Registry" [I-D.ietf-core-sid]:

```
SID range entry point: TBD
SID range size: 50
YANG module name: ietf-notification
reference: RFC-to-be
```

A ".sid" file is proposed in Appendix A.

8. Acknowledgements

The authors would like to thank Per Anderson, Andy Bierman, Carsten Bormann, Mohamed Boucadair, Tom Petch, Jason Sterne, Kent Watsen and Rob Wilton for their review and valuable comments.

9. References

9.1. Normative References

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-24, 22 December 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-sid-24>.

[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>.

[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>.

[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>.

[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>.

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

[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>.

[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>.

9.2. 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>.

[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>.

Appendix A. .sid file

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

For CBOR encoding using YANG-SIDs identifiers, a ".sid" file is requested to IANA in Section 7.3.

```
<CODE BEGINS> file "ietf-notification@2024-05-27.sid"
{
  "ietf-sid-file:sid-file": {
    "module-name": "ietf-notification",
    "module-revision": "2024-05-27",
    "description": "NETCONF Event Notification structure",
    "dependency-revision": [
      {
        "module-name": "ietf-yang-types",
        "module-revision": "2013-07-15"
      },
      {
        "module-name": "ietf-yang-structure-ext",
        "module-revision": "2020-06-17"
      }
    ],
    "assignment-range": [
      {
        "entry-point": "2550",
        "size": "50"
      }
    ],
    "item": [
      {
        "namespace": "module",
        "identifier": "ietf-notification",
        "sid": "2550"
      },
      {
        "namespace": "data",
        "identifier": "/ietf-notification:notification",
        "sid": "2551"
      },
      {
        "namespace": "data",
        "identifier": "/ietf-notification:notification/eventTime",
        "sid": "2552"
      }
    ]
  }
}
<CODE ENDS>
```

Figure 5: .sid file for "ietf-notification" module

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

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2024

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

1 March 2024

List Pagination Snapshots for YANG-driven Protocols
draft-awwhl-netconf-list-pagination-snapshot-00

Abstract

List pagination for YANG-driven protocols are defined in [I-D.ietf-netconf-list-pagination]. Operational data can have very large data sets. These data sets can furthermore have big churn, a lot of additions or deletions to the data set. In order to support a stable pagination of such data sets, snapshots can be used.

This document defines snapshot support for pagination of "config false" nodes of type "list" and "leaf-list". The snapshot support for individual nodes is signaled via the "ietf-system-capabilities" module.

Table of Contents

1. Introduction

The following open questions have been identified for list-pagination with snapshots.

The requirements that are necessory to resolve for a complete solution:

* What should be in the snapshot? The discussions have touched on include entire list content, take a snapshot of list keys etc.

* How should a client return to a taken snapshot? I.e. one RESTCONF request starts paginating and allocates a snapshot, how does the client return to that snapshot for the next page? The snapshot would need some id and a method to fetch it later. For instance a new query parameter to identify a snapshot, and a snapshot metadata id?

* What is the lifecycle of a snapshot for pagination?

* Should the client be able to signal that the snapshot should be deallocated?

* Should it the snapshot have some timeout after which it is deallocated?

* What happens when a server can’t take a snapshot due to resource constraints?

* Should snapshots be implicitly deallocated when the pagination reaches the last page?

* Security considerations for protecting against DoS when a lot of (possibly huge) snapshots can be taken.

1.1. Terminology

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

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

1.2. Conventions

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

1.3. Adherence to the NMDA

This document is compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. The "ietf-list-pagination-snapshot" module only defines a YANG identity, grouping, 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 extends the pagination functionality in [I-D.ietf-netconf-list-pagination]. The snapshot functionality defined by the document conforms to "config false" "list" and "leaf-list" nodes.

The "snapshot" query parameter (see Section 3) enables clients to ask create a snapshot. The support for snapshots is signaled via [RFC9196] (see Section 4).

3. The "snapshot" Query Parameter

Description
: The "snapshot" query parameter indicates that the client requests the server to take a snapshot of a "config false" target before starting the pagination.

Default Value
: If this query parameter is unspecified, it defaults to false.

Allowed Values
: The allowed values are true or false. If snapshots are not supported the "snapshot-not-supported" SHOULD be produced in the error-app-tag in the error output.

Conformance
: The "snapshot" query parameter MAY be supported for "config false" lists and leaf-lists.

3.1. NETCONF

For the NETCONF protocol, the "snapshot" query parameter is added to the protocol by augmenting "lpgsnap:snapshot-param-grouping" to the get, get-config, and get-data RPCs.

3.2. RESTCONF

The RESTCONF protocol specific functionality and conformance is defined in this section.

If the target node does not support snapshots, then a "501 Not Implemented" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value", and SHOULD also include the "snapshot-not-supported" identity as error-app-tag value.

Name	Methods	Description
snapshot	GET, HEAD	Indicates that the server should take a snapshot before paginating the result set.

The "snapshot" query parameter is allowed for GET and HEAD methods on "list" and "leaf-list" data resources. A "400 Bad Request" status-line MUST be returned if used with any other method or resource type. The error-tag value "operation-not-supprted" is used in this case.

4. Snapshot support

A server MAY support snapshots when paginating a "config false" list or leaf-list. In order to enable servers to identify which nodes may be used to take snapshots when paginating the "ietf-list-pagination-snapshot" module (see Section 5) augments an empty leaf node called "snapshot" into the "per-node-capabilities" node defined in the "ietf-system-capabilities" module (see [RFC9196]).

Note that it is possible for a client to request the server to take a snapshot when paginating with the "snapshot" query parameter (see Section 3.

5. The "ietf-list-pagination-snapshot" Module

The "ietf-list-pagination-snapshot" 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.

5.1. Data Model Overview

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

```
module: ietf-list-pagination-snapshot

  augment /nc:get/nc:input:
    +---w snapshot?   boolean
  augment /nc:get-config/nc:input:
    +---w snapshot?   boolean
  augment /ncds:get-data/ncds:input:
    +---w snapshot?   boolean
  augment /sysc:system-capabilities/sysc:datastore-capabilities
            /sysc:per-node-capabilities:
    +--ro snapshot?   empty
```

Comments:

As shown, this module augments an optional leaf into the "per-node-capabilities" list node of the "ietf-system-capabilities" module.

5.2. YANG Module

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

<CODE BEGINS> file "ietf-list-pagination-snapshot@2024-03-01.yang"

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

  import ietf-datastores {
    prefix ds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA)";
  }

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

  import ietf-netconf-nmda {
    prefix ncds;
```

```
    reference
      "RFC 8526: NETCONF Extensions to Support the
                 Network Management Datastore Architecture";
  }

  import ietf-system-capabilities  {
    prefix sysc;
    reference
      "RFC 9691: YANG Modules Describing Capabilities for Systems and
                 Datastore Update Notifications";
  }

  import ietf-list-pagination {
    prefix lpg;
    reference
      "draft-ietf-list-pagination: List Pagination for YANG-driven
                                   Protocols";
  }

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

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

  description
    "This module is used by servers to indicate they support
     snapshot pagination on 'config false' nodes of type 'list'
     and 'leaf-list'. It also defines a grouping for the snapshot
     parameter.

     Copyright (c) 2024 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Revised
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (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 2024-03-01 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: List Pagination Snapshots for YANG-driven
                   Protocols";
    }


    // Identities

    identity snapshot-not-supported {
      base lpg:list-pagination-error;
      description
        "Snapshot is not supported for the target. Either it is not a
         'config false' list or leaf-list, or it is disabled.";
    }

    // Groupings

    grouping snapshot-param-grouping {
      description
        "This grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      leaf snapshot {
        type boolean;
        description
          "The 'snapshot' parameter indicates that the client requests
           the server to take a snapshot of the 'config false' list or
           leaf-list target before paginating.";
      }
    }

    // Protocol-accessible nodes

    augment "/nc:get/nc:input" {
      description
        "Allow the 'get' operation to use the 'snapshot' query
         parameter for YANG list or leaf-list that is to be
         retrieved.";
      uses snapshot-param-grouping;
    }
```

```
    augment "/nc:get-config/nc:input" {
      description
        "Allow the 'get-config' operation to use the 'snapshot' query
         parameter for YANG list or leaf-list that is to be
         retrieved.";
      uses snapshot-param-grouping;
    }

    augment "/ncds:get-data/ncds:input" {
      description
        "Allow the 'get-data' operation to use the 'snapshot' query
         parameter for YANG list or leaf-list that is to be
         retrieved.";
      uses snapshot-param-grouping;
    }

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

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

      description
        "Defines some leafs that MAY be used by the server to
         describe constraints imposed of the 'where' filters and
         'sort-by' parameters used in list pagination queries.";
      leaf snapshot {
        type empty;
        description
          "Indicates that snapshots are supported for the targeted
           'config false' list or leaf-list node.";
      }
    }
  }

  <CODE ENDS>
```

6. IANA Considerations

6.1. The "IETF XML" Registry

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

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

6.2. The "YANG Module Names" Registry

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

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

6.3. The "RESTCONF Capability URNs" Registry

This document registers one capability in the RESTCONF Capability URNs [RFC8040] maintained at https://www.iana.org/assignments/restconf-capability-urns/restconf-capability-urns.xhtml. Following the instructions defined in Section 11.4 of [RFC8040], the below registrations are requested:

All the registrations are to use this document (RFC XXXX) for the "Reference" value.

Index	Capability Identifier
:snapshot	urn:ietf:params:restconf:capability:snapshot:1.0

7. Security Considerations

7.1. Regarding the "ietf-list-pagination-snapshot" YANG Module

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

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

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

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

8. References

8.1. Normative References

[I-D.ietf-netconf-list-pagination]
Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "List Pagination for YANG-driven Protocols", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-03, 1 March 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-netconf-list-pagination/>.

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

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

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

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

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

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

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

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

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

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

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

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

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

8.2. Informative References

[I-D.ietf-netconf-list-pagination-nc]
Watsen, K., Wu, Q., Hagsand, O., Li, H., and P. Andersson, "NETCONF Extensions to Support List Pagination", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-nc-02, 22 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-nc-02>.

[I-D.ietf-netconf-list-pagination-rc]
Watsen, K., Wu, Q., Hagsand, O., Li, H., and P. Andersson, "RESTCONF Extensions to Support List Pagination", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-rc-02, 22 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-rc-02>.

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

[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>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[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.

The vector tests assume the "example-social" YANG module and example data set defined [I-D.ietf-netconf-list-pagination].

A.1. Example Data Set

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

The following data enables snapshot support for the audit-log list node.

```
<system-capabilities
  xmlns="urn:ietf:params:xml:ns:yang:ietf-system-capabilities"
  xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores"
  xmlns:es="https://example.com/ns/example-social"
  xmlns:lpg="urn:ietf:params:xml:ns:yang:ietf-list-pagination">
  <datastore-capabilities>
    <datastore>ds:operational</datastore>
    <per-node-capabilities>
      <node-selector>/es:audit-logs/es:audit-log</node-selector>
      <lpgsnap:snapshot/>
    </per-node-capabilities>
  </datastore-capabilities>
</system-capabilities>
```

A.2. Example Queries

The following sections present example queries for the the snapshot query parameter.

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

A.2.1. The "snapshot" Parameter

The "snapshot" parameter may be used on "config false" target nodes.

> If this parameter is omitted, the default value is false.

REQUEST

```
Target: /example-social:audit-logs/audit-log
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     -
    Snapshot:  true
```

RESPONSE

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

Acknowledgements

The authors would like to thank the following for lively discussions on list (ordered by first name): Andy Bierman, Tom Petch, and Quifang Ma.

Authors' Addresses

Per Andersson
Cisco Systems
Email: perander@cisco.com

Kent Watsen
Watsen Networks

Email: kent+ietf@watsen.net

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

Olof Hagsand
SUNET
Email: olof@hagsand.se

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

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: 13 July 2024

J. Quilbeuf
B. Claise
Huawei
T. Graf
Swisscom
D. Lopez
Telefonica I+D
Q. Sun
China Telecom
10 January 2024

External Trace ID for Configuration Tracing
draft-ietf-netconf-configuration-tracing-00

Abstract

Network equipment are often configured by a variety of network management systems (NMS), protocols, and teams. If a network issue arises (e.g., because of a wrong configuration change), it is important to quickly identify the root cause and obtain the reason for pushing that modification. Another potential network issue can stem from concurrent NMSes with overlapping intents, each having their own tasks to perform. In such a case, it is important to map the respective modifications to its originating NMS.

This document specifies a NETCONF mechanism to automatically map the configuration modifications to their source, up to a specific NMS change request. Such a mechanism is required, in particular, for autonomous networks to trace the source of a particular configuration change that led to an anomaly detection. This mechanism facilitates the troubleshooting, the post mortem analysis, and in the end the closed loop automation required for self-healing networks. The specification also includes a YANG module that is meant to map a local configuration change to the corresponding trace id, up to the controller or even the orchestrator.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/JeanQuilbeufHuawei/draft-quilbeuf-opsawg-configuration-tracing.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

This specification focuses only on configuration pushed via NETCONF [RFC6241] or RESTCONF [RFC8040]. The rationale for this choice is that NETCONF is better suited for normalization than other protocols (SNMP, CLI). Another reason is that the notion of trace context, useful to track configuration modifications, has been ported to NETCONF in [I-D.rogaglia-netconf-trace-ctx-extension] and RESTCONF in [I-D.rogaglia-netconf-restconf-trace-ctx-headers].

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

This document specifies a mechanism to automatically map the configuration modifications to their source, up to a specific NMS service request. Practically, this mechanism annotates configuration changes on the configured element with sufficient information to unambiguously identify the corresponding transaction, if any, on the element that requested the configuration modification. It reuses the concept of Trace Context [W3C-Trace-Context] applied to NETCONF as in [I-D.ietf-netconf-transaction-id] The information needed to trace the configuration is stored in a new YANG module that maps a local configuration change to some additional metadata. The additional metadata contains the trace ID, and, if the local change is not the beginning of the trace, the ID of the client that triggered the local-change.

2. Terminology

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

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

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

This document uses the term trace ID from [W3C-Trace-Context].

Local Commit ID Identifier of a local configuration change on a Network Equipment, Controller, Orchestrator or any other device or software handling configuration. Such an identifier is usually present in devices that can show an history of the configuration changes, to identify one such configuration change.

3. Use cases

This document was written with autonomous networks in mind. We assume that an existing monitoring or assurance system, such as described in [RFC9417], is able to detect and report network anomalies , e.g. SLA violations, intent violations, network failure, or simply a customer issue. Here are the use cases for the proposed YANG module.

3.1. Configuration Mistakes

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

3.2. Concurrent NMS Configuration

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

3.3. Conflicting Intents

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

3.4. Not a use case: Onboarding

During onboarding, a newly added device is likely to receive a multiple configuration message, as it needs to be fully configured. Our use cases focus more on what happens after the initial configuration is done, i.e. when the "stable" configuration is modified.

4. Relying on W3C Trace Context to Trace Configuration Modifications

4.1. Existing configuration metadata on device

This document assumes that NETCONF clients or servers (orchestrators, controllers, devices, ...) have some kind of mechanism to record the modifications done to the configuration. For instance, devices typically have an history of configuration changes and this history associates a locally unique identifier to some metadata, such as the timestamp of the modification, the user doing the modification or the protocol used for the modification. Such a locally unique identifier is a Local Commit ID, we assume that it exists on the platform. This Local Commit ID is the link between the module presented in this draft and the device-specific way of storing configuration changes.

4.2. Client ID

This document assumes that each NETCONF client for which configuration must be traced (for instance orchestrator and controllers) has a unique client ID among the other NETCONF clients in the network. Such an ID could be an IP address or a host name. The mechanism for providing and defining this client ID is out of scope of the current document.

4.3. Instantiating the YANG module

[I-D.rogaglia-netconf-trace-ctx-extension] defines a NETCONF extension providing the trace context from [W3C-Trace-Context]. Using this mechanism, the NETCONF server captures the trace-id, when available, and maps it to a local commit ID, by populating the YANG module.

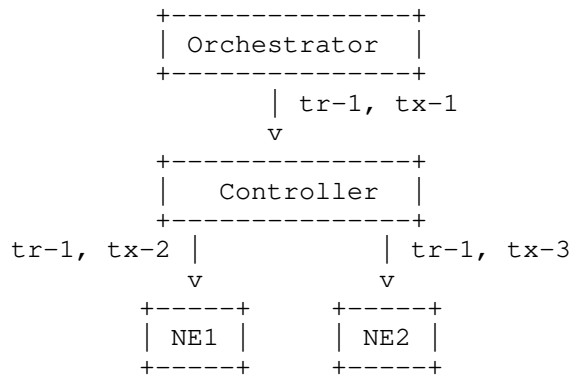


Figure 1: Example of Hierarchical Configuration. tx: transaction. tr: trace.

It is technically possible that several clients push configuration to the candidate configuration datastore and only one of them commits the changes to the running configuration datastore. From the running configuration datastore perspective, which is the effective one, there is a single modification, but caused by several clients, which means that this modification should have several corresponding client-ids. Although, this case is technically possible, it is a bad practice. We wont cover it in this document. In other terms, we assume that a given configuration modification on a server is caused by a single client, and thus has a single corresponding client-id.

4.4. Using the YANG module

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

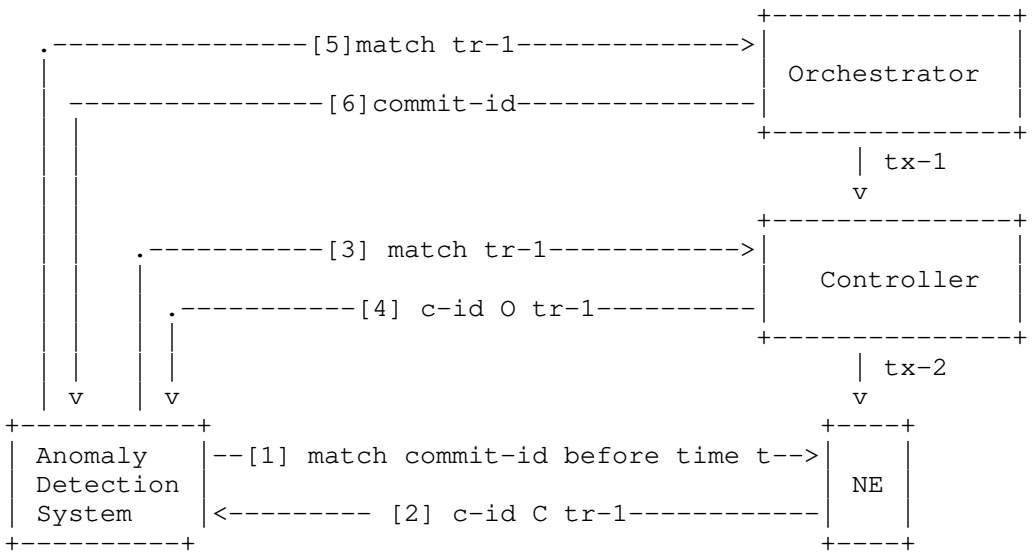


Figure 2: Example of Configuration Tracing. tr: trace-id, C: Controller, O: orchestrator. The number between square brackets refer to steps in the listing below.

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

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

2. The ADS receives the trace-id and the client-id. In Figure 2, that step would receive the trace-id tr-1 and the id of the Controller as a result. If there is no associated client-id, the change was not done by a client compatible with the present draft, and the investigation stops here.

3. The ADS queries the client identified by the client-id found at the previous step, looking for a match of the trace-id from the previous step. In Figure 2, for that step, the software would look for the trace-id tr-1 stored in the Controller.

4. From that query, the ADS knows the local-commit-id on the client (Controller in our case). Since the local-commit-id is associated to a client-id pointing to the Orchestrator, the ADS continues the investigation.

5. The ADS queries the Orchestrator, trying to find a match for the trace-id tr-1.

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

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

5. YANG module

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

5.1. Overview

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

```
module: ietf-external-transaction-id
  +--ro external-transactions-id
     +--ro configuration-change* [local-commit-id]
        +--ro local-commit-id    string
        +--ro timestamp?         yang:date-and-time
        +--ro trace-parent
        |  +--ro version?       hex-digits
        |  +--ro trace-id?      hex-digits
        |  +--ro parent-id?     hex-digits
        |  +--ro trace-flags?   hex-digits
        +--ro client-id?         string
```

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

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

The trace-parent is present to identify the trace associated to the local-commit-id. This trace-parent can be transmitted by a client or created by the current server. In Section 4.4, the most important field in trace-parent is the trace-id. We also included the other fields for trace-parent as defined in [W3C-Trace-Context] for the sake of completion. In some cases, for instance direct configuration of the device, the device may choose to not include the trace-id.

The presence of a client-id indicates that the trace-parent has been transmitted by that client. If the trace is initiated by the current server, there is no associated client-id.

Even if this document focuses only on NETCONF or RESTCONF, the use cases defined in Section 3 are not specific to NETCONF or RESTCONF and the mechanism described in this document could be adapted to other configuration mechanisms. For instance, a configuration modification pushed via CLI can be identified via a label, which could contain the trace-parent. As such cases are difficult to standardize, we wont cover them in this document.

5.2. YANG module ietf-external-transaction-id

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

```
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types, Section 3";
  }

  organization
    "IETF NETCONF Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>
     Author:   Benoit Claise  <mailto:benoit.claise@huawei.com>
     Author:   Jean Quilbeuf  <mailto:jean.quilbeuf@huawei.com>";
  description
    "This module enables tracing of configuration changes in a
     network for the sake of automated correlation between
     configuration changes and the external request that triggered
     that change.

     The module stores the identifier of the trace, if any, that
     triggered the change in a device. If that trace-id was provided
     by a client, (i.e. not created locally by the server), the id
     of that client is stored as well to indicated which client
     triggered the configuration change.

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

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

  revision 2022-10-20 {
    description
      "Initial revision";
    reference
      "RFC xxxx: Title to be completed";
  }

  typedef hex-digits {
    type string {
      pattern '[0-9a-f]*';
    }
```

```
      description
        "A string composed of hexadecimal digits. Digits represented by
         letters are restricted to lowercase so that a single
         representation of a given value is allowed. This enables using
         the string equality to check equality of the represented
         values.";
    }

    grouping trace-parent-g {
      description
        "Trace parent frow the W3C trace-context recommandation.
         Follows the format version 00.";
      leaf version {
        type hex-digits {
          length "2";
        }
        must "../version = '00'";
        description
          "Version of the trace context. Must be 00 to match the
           format described in this module.";
      }
      leaf trace-id {
        type hex-digits {
          length "32";
        }
        must "../trace-id != '00000000000000000000000000000000'";
        description
          "Trace ID that is common for every transaction that is
           part of the configuration chain. This value can be used
           to match a local commit id to a commit local to another
           system.";
      }
      leaf parent-id {
        type hex-digits {
          length "16";
        }
        description
          "ID of the request (client-side) that lead to configuring
           the server hosting this module.";
      }
      leaf trace-flags {
        type hex-digits {
          length "2";
        }
        description
          "Flags enabled for this trace. See W3C reference for the
           details about flags.";
      }
```

```
    }

    container external-transactions-id {
      config false;
      description
        "Contains the IDs of configuration transactions that are
         external to the device.";
      list configuration-change {
        key "local-commit-id";
        description
          "List of configuration changes, identified by their
           local-commit-id";
        leaf local-commit-id {
          type string;
          description
            "Stores the identifier as saved by the server. Can be used
             to retrieve the corresponding changes using the server
             mechanism if available.";
        }
        leaf timestamp {
          type yang:date-and-time;
          description
            "A timestamp that can be used to further filter change
             events.";
        }
        container trace-parent {
          description
            "Trace parent associated to the local-commit-id. If a
             client ID is present as well, the trace context was
             transmitted by that client. If not, the trace context was
             created locally.

             This trace-parent must come from the trace context of the
             request actually modifying the running configuration
             datastore. This request might be an edit-config or a
             commit depending on whether the candidate datastore is
             used.";
          uses trace-parent-g;
        }
        leaf client-id {
          type string;
          description
            "ID of the client that originated the modification, to
             further query information about the corresponding
             change.

             This data node is present only when the configuration was
             pushed by a compatible system.";
```

```
        }
      }
    }
  }
  <CODE ENDS>
```

6. Security Considerations

7. IANA Considerations

This document includes no request to IANA.

8. Contributors

9. Open Issues / TODO

* Indicate what to do with O-RAN apps, since each of them might be seen as a different client with a different client-id. This is actually a requirement that the client-id should be granular enough to distinguish between different controllers colocated on the same device. For instance, the IP address might not be a suitable client-id in that case.

* Define how to pass the client-id. Current leads are the trace-state from [W3C-Trace-Context] and W3C Baggage (https://www.w3.org/TR/baggage/).

* The model and usage presented here focuses of the problem of tracing a configuration change back to its sources. As it relies on [W3C-Trace-Context], we could also use associated mechanisms for collecting and representing trace data such as OTLP. For instance, we could define a YANG model matching the OTLP protobuffer definition (draft: https://github.com/rgaglian/ietf-netconf-trace-context-extension/blob/main/ietf-netconf-otlp-protocol.tree). In that case the client-id could be a specific attribute of the spans list.

10. Normative References

[I-D.ietf-netconf-transaction-id]
Lindblad, J., "Transaction ID Mechanism for NETCONF", Work in Progress, Internet-Draft, draft-ietf-netconf-transaction-id-01, 4 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-transaction-id-01>.

[I-D.rogaglia-netconf-restconf-trace-ctx-headers]
Gagliano, R., Larsson, K., and J. Lindblad, "RESTCONF Extension to support Trace Context Headers", Work in Progress, Internet-Draft, draft-rogaglia-netconf-restconf-trace-ctx-headers-00, 6 July 2023, <https://datatracker.ietf.org/doc/html/draft-rogaglia-netconf-restconf-trace-ctx-headers-00>.

[I-D.rogaglia-netconf-trace-ctx-extension]
Gagliano, R., Larsson, K., and J. Lindblad, "NETCONF Extension to support Trace Context propagation", Work in Progress, Internet-Draft, draft-rogaglia-netconf-trace-ctx-extension-03, 6 July 2023, <https://datatracker.ietf.org/doc/html/draft-rogaglia-netconf-trace-ctx-extension-03>.

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

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

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

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

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

11. Informative References

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

[RFC9417] Claise, B., Quilbeuf, J., Lopez, D., Voyer, D., and T. Arumugam, "Service Assurance for Intent-Based Networking Architecture", RFC 9417, DOI 10.17487/RFC9417, July 2023, <https://www.rfc-editor.org/info/rfc9417>.

Appendix A. Changes between revisions

01 -> 02

* Switch to trace-parent instead of transaction id for tracing configuration

00 -> 01

* Define Parent and Child Transaction

* Context for the "local-commit-id" concept

* Feedback from Med, both in text and YANG module

Appendix B. Tracing configuration changes

Acknowledgements

The authors would like to thank Mohamed Boucadair, Jan Linblad and Roque Gagliano for their reviews and propositions.

Authors' Addresses

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

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

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

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

Qiong Sun
China Telecom
Email: sunqiong@chinatelecom.cn

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2025

J. Quilbeuf
B. Claise
Huawei
T. Graf
Swisscom
D. Lopez
Telefonica I+D
Q. Sun
China Telecom
21 October 2024

External Trace ID for Configuration Tracing
draft-ietf-netconf-configuration-tracing-03

Abstract

Network equipment are often configured by a variety of network management systems (NMS), protocols, and teams. If a network issue arises (e.g., because of a wrong configuration change), it is important to quickly identify the root cause and obtain the reason for pushing that modification. Another potential network issue can stem from concurrent NMSes with overlapping intents, each having their own tasks to perform. In such a case, it is important to map the respective modifications to its originating NMS.

This document specifies a NETCONF mechanism to automatically map the configuration modifications to their source, up to a specific NMS change request. Such a mechanism is required, in particular, for autonomous networks to trace the source of a particular configuration change that led to an anomaly detection. This mechanism facilitates the troubleshooting, the post-mortem analysis, and in the end the closed loop automation required for self-healing networks. The specification also includes a YANG module that is meant to map a local configuration change to the corresponding trace id, up to the controller or even the orchestrator.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/JeanQuilbeufHuawei/draft-quilbeuf-opsawg-configuration-tracing.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

This specification focuses only on configuration pushed via NETCONF [RFC6241] or RESTCONF [RFC8040]. The rationale for this choice is that NETCONF is better suited for normalization than other protocols (SNMP, CLI). Another reason is that the notion of trace context, useful to track configuration modifications, has been ported to NETCONF in [I-D.ietf-netconf-trace-ctx-extension] and RESTCONF in [I-D.ietf-netconf-restconf-trace-ctx-headers].

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

This document specifies a mechanism to automatically map the configuration modifications to their source, up to a specific NMS service request. Practically, this mechanism annotates configuration changes on the configured element with sufficient information to unambiguously identify the corresponding transaction, if any, on the element that requested the configuration modification. It reuses the concept of Trace Context [W3C-Trace-Context] applied to NETCONF as in [I-D.ietf-netconf-trace-ctx-extension] The information needed to trace the configuration is stored in a new YANG module that maps a local configuration change to some additional metadata. The additional metadata contains the trace ID, and, if the local change is not the beginning of the trace, the ID of the client that triggered the local-change. In that sense, it is an instance of the YANG DataStore implementation of the Trace Context as proposed in Section 1.2 of [I-D.ietf-netconf-trace-ctx-extension].

2. Terminology

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

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

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

This document uses the term trace ID from [W3C-Trace-Context].

Local Commit ID Identifier of a local configuration change on a Network Equipment, Controller, Orchestrator or any other device or software handling configuration. Such an identifier is usually present in devices that can show a history of the configuration changes, to identify one such configuration change.

3. Use cases

This document was written with autonomous networks in mind. We assume that an existing monitoring or assurance system, such as described in [RFC9417], is able to detect and report network anomalies , e.g. SLA violations, intent violations, network failure, or simply a customer issue. Here are the use cases for the proposed YANG module; they are extensions of the "Provisioning root cause analysis" use case presented in Section 1.3.1 of [I-D.ietf-netconf-trace-ctx-extension].

3.1. Configuration Mistakes

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

3.2. Concurrent NMS Configuration

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

3.3. Conflicting Intents

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

3.4. Not a use case: Onboarding

During onboarding, a newly added device is likely to receive a multiple configuration message, as it needs to be fully configured. Our use cases focus more on what happens after the initial configuration is done, i.e. when the "stable" configuration is modified.

4. Relying on W3C Trace Context to Trace Configuration Modifications

4.1. Existing configuration metadata on device

This document assumes that NETCONF clients or servers (orchestrators, controllers, devices, ...) have some kind of mechanism to record the modifications done to the configuration. For instance, devices typically have a history of configuration changes and this history associates a locally unique identifier to some metadata, such as the timestamp of the modification, the user doing the modification or the protocol used for the modification. Such a locally unique identifier is a Local Commit ID, we assume that it exists on the platform. This Local Commit ID is the link between the module presented in this draft and the device-specific way of storing configuration changes.

4.2. Client ID

This document assumes that each NETCONF client for which configuration must be traced (for instance orchestrator and controllers) has a unique client ID among the other NETCONF clients in the network. Such an ID could be an IP address or a host name. The mechanism for providing and defining this client ID is out of scope of the current document.

4.3. Instantiating the YANG module

[I-D.ietf-netconf-trace-ctx-extension] defines a NETCONF extension providing the trace context from [W3C-Trace-Context]. Using this mechanism, the NETCONF server captures the trace-id, when available, and maps it to a local commit ID, by populating the YANG module.

The trace context from W3C provides a parent-id. This parent-id does not identify a particular server or NMS but rather one request in the chain of HTTP requests constituting the trace. Similarly to the passing of the trace context in [I-D.ietf-netconf-trace-ctx-extension], we propose an XML attribute on NETCONF messages to pass the client-id. The attribute name is "client-id" and the namespace is the namespace of the YANG module from Section 5, namely 'urn:ietf:params:xml:ns:yang:ietf-external-transaction-id'. An example of a commit message including the client-id is shown in Figure 4.

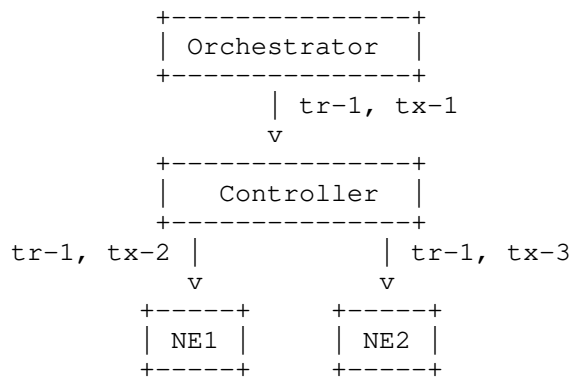


Figure 1: Example of Hierarchical Configuration. tx: transaction. tr: trace.

In Figure 1, the transactions 'tx-1', 'tx-2' and 'tx-3' are sent via NETCONF. The NETCONF RPC used, most likely 'commit' in our use case, is annotated with the 'traceparent' annotation as defined in [I-D.ietf-netconf-trace-ctx-extension]. The traceparent annotation has the same trace id 'tr-1' for each of these transactions. Additionally, for each transaction the client id is passed via the 'client-id' annotation. For 'tx-1' the client-id is the id of the Orchestrator. For 'tx-2' and 'tx-3', the client is the id of the Controller.

It is technically possible that several clients push configuration to the candidate configuration datastore and only one of them commits the changes to the running configuration datastore. From the running configuration datastore perspective, which is the effective one, there is a single modification, but caused by several clients, which means that this modification should have several corresponding client-ids. Although, this case is technically possible, it is a bad practice. We wonâ\200\231t cover it in this document. In other terms, we assume that a given configuration modification on a server is caused by a single client, and thus has a single corresponding client-id.

4.4. Using the YANG module

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

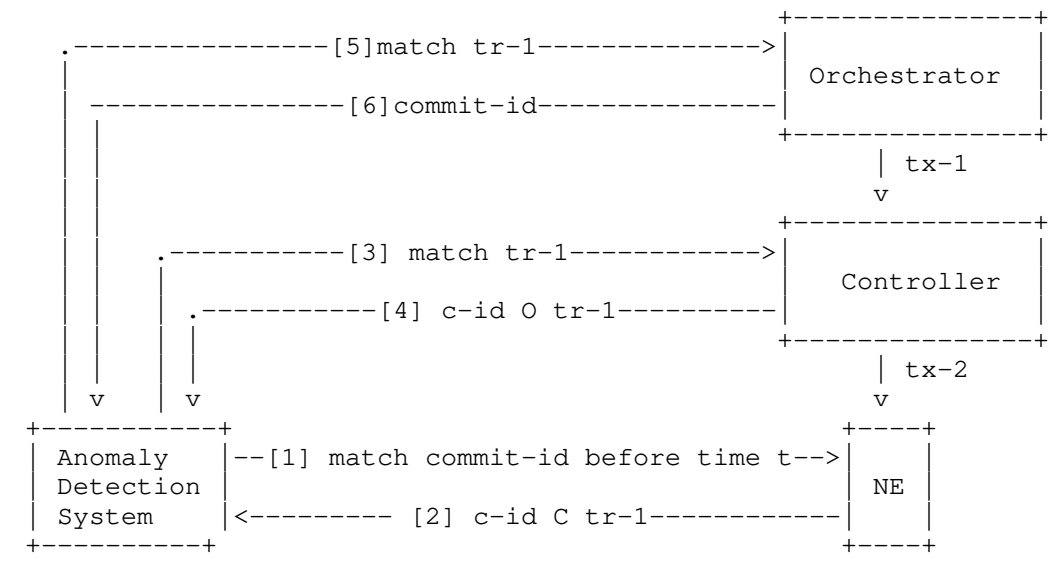


Figure 2: Example of Configuration Tracing. tr: trace-id, C: Controller, O: orchestrator. The number between square brackets refer to steps in the listing below.

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

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

2. The ADS receives the trace-id and the client-id. In Figure 2, that step would receive the trace-id tr-1 and the id of the Controller as a result. If there is no associated client-id, the change was not done by a client compatible with the present draft, and the investigation stops here.

3. The ADS queries the client identified by the client-id found at the previous step, looking for a match of the trace-id from the previous step. In Figure 2, for that step, the software would look for the trace-id tr-1 stored in the Controller.

4. From that query, the ADS knows the local-commit-id on the client (Controller in our case). Since the local-commit-id is associated to a client-id pointing to the Orchestrator, the ADS continues the investigation.

5. The ADS queries the Orchestrator, trying to find a match for the trace-id tr-1.

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

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

5. YANG module

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

5.1. Overview

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

```
module: ietf-external-transaction-id
  +--ro external-transactions-id
     +--ro configuration-change* [local-commit-id]
        +--ro local-commit-id    string
        +--ro timestamp?         yang:date-and-time
        +--ro trace-parent
        |  +--ro version?       hex-digits
        |  +--ro trace-id?      hex-digits
        |  +--ro parent-id?     hex-digits
        |  +--ro trace-flags?   hex-digits
        +--ro client-id?         string
```

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

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

The trace-parent is present to identify the trace associated to the local-commit-id. This trace-parent can be transmitted by a client or created by the current server. In Section 4.4, the most important field in trace-parent is the trace-id. We also included the other fields for trace-parent as defined in [W3C-Trace-Context] for the sake of completion. In some cases, for instance direct configuration of the device, the device may choose to not include the trace-id.

The presence of a client-id indicates that the trace-parent has been transmitted by that client. If the trace is initiated by the current server, there is no associated client-id.

Even if this document focuses only on NETCONF or RESTCONF, the use cases defined in Section 3 are not specific to NETCONF or RESTCONF and the mechanism described in this document could be adapted to other configuration mechanisms. For instance, a configuration modification pushed via CLI can be identified via a label, which could contain the trace-parent. As such cases are difficult to standardize, we wonâ\200\231t cover them in this document.

5.2. YANG module ietf-external-transaction-id

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

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types, Section 3";
  }

  organization
    "IETF NETCONF Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>
     Author:   Benoit Claise  <mailto:benoit.claise@huawei.com>
     Author:   Jean Quilbeuf  <mailto:jean.quilbeuf@huawei.com>";
  description
    "This module enables tracing of configuration changes in a
     network for the sake of automated correlation between
     configuration changes and the external request that triggered
     that change.
```

```
        The module stores the identifier of the trace, if any, that
        triggered the change in a device. If that trace-id was provided
        by a client, (i.e. not created locally by the server), the id
        of that client is stored as well to indicated which client
        triggered the configuration change.

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

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

    revision 2022-10-20 {
      description
        "Initial revision";
      reference
        "RFC xxxx: Title to be completed";
    }

    typedef hex-digits {
      type string {
        pattern '[0-9a-f]*';
      }
      description
        "A string composed of hexadecimal digits. Digits represented by
         letters are restricted to lowercase so that a single
         representation of a given value is allowed. This enables using
         the string equality to check equality of the represented
         values.";
    }

    grouping trace-parent-g {
      description
        "Trace parent frow the W3C trace-context recommandation.
         Follows the format version 00.";
      leaf version {
        type hex-digits {
          length "2";
        }
        must "../version = '00'";
        description
          "Version of the trace context. Must be 00 to match the
```

```
            format described in this module.";
        }
        leaf trace-id {
          type hex-digits {
            length "32";
          }
          must "../trace-id != '00000000000000000000000000000000'";
          description
            "Trace ID that is common for every transaction that is
             part of the configuration chain. This value can be used
             to match a local commit id to a commit local to another
             system.";
        }
        leaf parent-id {
          type hex-digits {
            length "16";
          }
          description
            "ID of the request (client-side) that lead to configuring
             the server hosting this module.";
        }
        leaf trace-flags {
          type hex-digits {
            length "2";
          }
          description
            "Flags enabled for this trace. See W3C reference for the
             details about flags.";
        }
      }

      container external-transactions-id {
        config false;
        description
          "Contains the IDs of configuration transactions that are
           external to the device.";
        list configuration-change {
          key "local-commit-id";
          description
            "List of configuration changes, identified by their
             local-commit-id";
          leaf local-commit-id {
            type string;
            description
              "Stores the identifier as saved by the server. Can be used
               to retrieve the corresponding changes using the server
               mechanism if available.";
          }
```

```
        leaf timestamp {
          type yang:date-and-time;
          description
            "A timestamp that can be used to further filter change
             events.";
        }
        container trace-parent {
          description
            "Trace parent associated to the local-commit-id. If a
             client ID is present as well, the trace context was
             transmitted by that client. If not, the trace context was
             created locally.

             This trace-parent must come from the trace context of the
             request actually modifying the running configuration
             datastore. This request might be an edit-config or a
             commit depending on whether the candidate datastore is
             used.";
          uses trace-parent-g;
        }
        leaf client-id {
          type string;
          description
            "ID of the client that originated the modification, to
             further query information about the corresponding
             change.

             This data node is present only when the configuration was
             pushed by a compatible system.";
        }
      }
    }
  }
  <CODE ENDS>
```

6. Security Considerations

7. IANA Considerations

This document includes no request to IANA.

8. Contributors

9. Open Issues / TODO

* The model and usage presented here focuses of the problem of tracing a configuration change back to its sources. As it relies on [W3C-Trace-Context], we could also use associated mechanisms

for collecting and representing trace data such as OTLP. For instance, we could define a YANG model matching the OTLP protobuffer definition (draft: https://github.com/rgaglian/ietf-netconf-trace-context-extension/blob/main/ietf-netconf-otlp-protocol.tree). In that case the client-id could be a specific attribute of the spans list.

10. Normative References

[I-D.ietf-netconf-restconf-trace-ctx-headers]
Gagliano, R., Larsson, K., and J. Lindblad, "RESTCONF Extension to Support Trace Context Headers", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-trace-ctx-headers-02, 25 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-restconf-trace-ctx-headers-02>.

[I-D.ietf-netconf-trace-ctx-extension]
Gagliano, R., Larsson, K., and J. Lindblad, "NETCONF Extension to support Trace Context propagation", Work in Progress, Internet-Draft, draft-ietf-netconf-trace-ctx-extension-01, 8 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-trace-ctx-extension-01>.

[I-D.ietf-netconf-transaction-id]
Lindblad, J., "Transaction ID Mechanism for NETCONF", Work in Progress, Internet-Draft, draft-ietf-netconf-transaction-id-07, 19 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-transaction-id-07>.

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

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

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

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

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

11. Informative References

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

[RFC9417] Claise, B., Quilbeuf, J., Lopez, D., Voyer, D., and T. Arumugam, "Service Assurance for Intent-Based Networking Architecture", RFC 9417, DOI 10.17487/RFC9417, July 2023, <https://www.rfc-editor.org/info/rfc9417>.

Appendix A. Changes between revisions

01 -> 02

* Remove YANG specific annotation for the mechanism to pass the client-id.

* Align with NETCONF Trace context draft.

00(WG adoption) -> 01

* Define mechanism to pass the client-id.

01 -> 02

* Switch to trace-parent instead of transaction id for tracing configuration

00 -> 01

* Define Parent and Child Transaction

* Context for the "local-commit-id" concept

* Feedback from Med, both in text and YANG module

Appendix B. Example of NETCONF message

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1"
     xmlns:w3ctc="urn:ietf:params:xml:ns:netconf:w3ctc:1.0"
     xmlns:ext-txid="urn:ietf:params:xml:ns:yang:ietf-external-transaction-id"
     w3ctc:traceparent=
       "00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01"
     ext-txid:client-id="controller-01">
  <commit/>
</rpc>
```

Figure 4: Example of NETCONF commit RPC with annotations

In Figure 4, we present an RPC annotated with the traceparent and the client-id. The traceparent example is taken from [I-D.ietf-netconf-trace-ctx-extension]. The client-id annotation is defined in our YANG module. Here the client-id passed is 'controller-01'.

Acknowledgements

The authors would like to thank Mohamed Boucadair, Jan Linblad and Roque Gagliano for their reviews and propositions.

Authors' Addresses

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

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

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

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 82
Madrid 28006
Spain

Email: diego.r.lopez@telefonica.com

Qiong Sun
China Telecom
Email: sunqiong@chinatelecom.cn

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2024

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

1 March 2024

List Pagination for YANG-driven Protocols
draft-ietf-netconf-list-pagination-03

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

1.3. Adherence to the NMDA

This document is compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. The "ietf-list-pagination" module 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 3.1.4), and either a combination of the "offset" parameter (see Section 3.1.5) or the "cursor" parameter (see Section 3.1.6), and lastly "the "limit" parameter (see Section 3.1.7).

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

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 "locale" Query Parameter

Description
: The "locale" query parameter indicates what locale is used when collating the result-set.

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

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

Conformance
: The "locale" 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.

3.1.4. The "direction" Query Parameter

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

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

Allowed Values
: The allowed values are:

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

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

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

3.1.5. The "offset" Query Parameter

Description
: The "offset" query parameter indicates the number of entries in the working result-set (i.e., after the "direction" parameter has been applied) that should be skipped over when preparing the response.

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-app-tag in the error output when this occurs.

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

3.1.6. The "cursor" Query Parameter

Description
: The "cursor" query parameter indicates where to start the working result-set (i.e., after the "direction" parameter has been applied), the elements before the cursor are skipped over when preparing the response. Furthermore, a result set constrained with the "limit" query parameter includes metadata values [RFC7952] called "next" and "previous", which contains cursor values to the next and previous result-sets. These next and previous cursor values are opaque index values for the underlying system's database, e.g. a key or other information needed to

efficiently access the selected result-set. These "next" and "previous" metadata values work as Hypermedia as the Engine of Application State (HATEOAS) links [REST-Dissertation]. This means that the server does not keep any stateful information about the "next" and "previous" cursor or the current page. Due to their ephemeral nature, cursor values are never cached.

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, e.g. a list key or other information to efficiently access the selected result-set. 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-app-tag in the error output when this occurs.

Conformance
The "cursor" query parameter MUST be supported for all "config true" lists and SHOULD be supported for all "config false" lists. It is however optional to support the "cursor" query parameter for "config false" lists and the support must be signaled by the server per list.

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

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

3.1.7. The "limit" Query Parameter

Description
The "limit" query parameter limits the number of entries returned from the working result-set (i.e., after the "offset" parameter has been applied). Any list or leaf-list that is limited includes, somewhere in its encoding, a metadata value [RFC7952] called "remaining", a positive integer indicating the number of elements that were not included in the result-set by the "limit" operation, or the value "unknown" in case, e.g., the server

determines that counting would be prohibitively expensive.

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 [RFC9196].

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 [RFC9196]).

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
    +--ro cursor-supported?   empty
```

Comments:

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

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

4.2. Example Usage

4.2.1. Constraining a "config false" list

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

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

   <system-capabilities
     xmlns="urn:ietf:params:xml:ns:yang:ietf-system-capabilities"
     xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores"
     xmlns:es="https://example.com/ns/example-social"
     xmlns:lpg="urn:ietf:params:xml:ns:yang:ietf-list-pagination">
     <datastore-capabilities>
       <datastore>ds:operational</datastore>
       <per-node-capabilities>
         <node-selector>/es:audit-logs/es:audit-log</node-selector>
         <lpg:constrained/>
       </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 [RFC9196].

```
   <CODE BEGINS> file "ietf-list-pagination@2024-03-01.yang"

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

     import ietf-datastores {
```

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

    import ietf-yang-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
        "RFC 9196: 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) 2024 IETF Trust and the persons identified
       as authors of the code. All rights reserved.

       Redistribution and use in source and binary forms, with
       or without modification, is permitted pursuant to, and
       subject to the license terms contained in, the Revised
       BSD License set forth in Section 4.c of the IETF Trust's
       Legal Provisions Relating to IETF Documents
       (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 2024-03-01 {
      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.";
    }

    md:annotation next {
      type string;
      description
        "This annotation contains the base64 encoded value of the next
```

```
         cursor in the pagination.";
    }

    md:annotation previous {
      type string;
      description
        "This annotation contains the base64 encoded value of the
         previous cursor in the pagination.";
    }

    md:annotation locale {
      type string;
      description
        "This annotation contains the locale used when sorting.

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

         For further details see references:
         RFC 5646: Tags for identifying Languages
         RFC 6365: Technology Used in Internationalization in the
                   IETF";
    }

    // Identities

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

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

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

    identity locale-unavailable {
      base list-pagination-error;
```

```
      description
        "The 'locale' query parameter input is not a valid
         locale or the locale is not available on the system.";
    }

    // Groupings

    grouping 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 locale-param-grouping {
      description
        "The grouping may be used by protocol-specific YANG modules
         to define a protocol-specific query parameter.";
      leaf locale {
        type string;
        description
          "The 'locale' parameter indicates the locale which the
           entries in the working result-set should be collated.";
      }
    }
```

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

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

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

      leaf constrained {
        type 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.";
      }
      leaf cursor-supported {
        type empty;
        description
          "Indicates that the targeted list node supports the
           'cursor' parameter.";
      }
    }
  }

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

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

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

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

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

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

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

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

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>.

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

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

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

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

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

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

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

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

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

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

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

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

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

7.2. Informative References

[I-D.ietf-netconf-list-pagination-nc]
Watsen, K., Wu, Q., Hagsand, O., Li, H., and P. Andersson, "NETCONF Extensions to Support List Pagination", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-nc-02, 22 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-nc-02>.

[I-D.ietf-netconf-list-pagination-rc]
Watsen, K., Wu, Q., Hagsand, O., Li, H., and P. Andersson, "RESTCONF Extensions to Support List Pagination", Work in

Progress, Internet-Draft, draft-ietf-netconf-list-pagination-rc-02, 22 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-rc-02>.

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

[REST-Dissertation]
Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[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>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <https://www.rfc-editor.org/info/rfc6365>.

[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 "https://example.com/ns/example-social";
  prefix es;

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

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

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

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

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

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

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

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

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

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

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

        container privacy-settings {
          leaf hide-network {
            type boolean;
            description
              "Hide who you follow and who follows you.";
          }
          leaf post-visibility {
            type enumeration {
              enum public {
                description
                  "Posts are public.";
              }
              enum unlisted {
                description
                  "Posts are unlisted, though 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"
          }
        },
        {
          "member-id": "åsa",
          "email-address": "asa@example.com",
          "password": "$0$1543",
          "avatar": "BASE64VALUE=",
          "privacy-settings": {
            "post-visibility": "unlisted"
          },
          "following": ["alice", "bob"],
          "stats": {
            "joined": "2022-02-19T13:12:00Z",
            "membership-level": "standard",
            "last-activity": "2022-04-19T13:12:59Z"
          }
        }
      ]
    },
```

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

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

A.3. Example Queries

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

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

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 "locale" Parameter

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

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

REQUEST

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

RESPONSE

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

```
REQUEST

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

RESPONSE
```

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

A.3.8. The "sublist-limit" Parameter

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

A.3.8.1. target is a list entry

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

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

This example sets the sublist-limit value '1', which returns just the first entry for all 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.3.8.2. 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.3.9. Combinations of Parameters

A.3.9.1. All six parameters at once

REQUEST

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

   RESPONSE

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

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

Acknowledgements

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

Per Andersson
Cisco Systems
Email: perander@cisco.com

Olof Hagsand
SUNET
Email: olof@hagsand.se

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

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2025

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

21 October 2024

List Pagination for YANG-driven Protocols
draft-ietf-netconf-list-pagination-05

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]. How the NETCONF and RESTCONF protocols support list pagination is described in [I-D.ietf-netconf-list-pagination-nc] and [I-D.ietf-netconf-list-pagination-rc], respectively.

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

1.1. Terminology

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

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

1.2. Conventions

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

1.3. Adherence to the NMDA

This document is compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. The "ietf-list-pagination" module 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.

Furthermore the solution is intended to leverage underlying database system capabilities, e.g. fast lookups relying on indexes, using efficient built-in selection and pagination functions. However, since there are many different database systems and configurations, the solution defines a common subset of functionality broadly available. It is also possible that a datastore's underlying database system is federated, i.e. consists of several different database systems to provide one datastore. In order to form a general solution, the possibility to configure and tune what components are available is presented.

A secondary aspect of this solution entails a client sending a query parameter to a server guiding how 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 3.1.4), and either a combination of the "offset" parameter (see Section 3.1.5) or the "cursor" parameter (see Section 3.1.6), and lastly "the "limit" parameter (see Section 3.1.7).

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

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. The XPath context follows Section 6.4.1 of [RFC7950] and details (such as prefix bindings) are further defined at the protocol level, see [I-D.ietf-netconf-list-pagination-nc] and [I-D.ietf-netconf-list-pagination-rc]. 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 "locale" Query Parameter

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

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

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

Conformance
The "locale" 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.

3.1.4. The "direction" Query Parameter

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

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

Allowed Values
: The allowed values are:

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

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

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

3.1.5. The "offset" Query Parameter

Description
: The "offset" query parameter indicates the number of entries in the working result-set (i.e., after the "direction" parameter has been applied) that should be skipped over when preparing the response.

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-app-tag in the error output when this occurs.

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

3.1.6. The "cursor" Query Parameter

Description
The "cursor" query parameter indicates where to start the working result-set (i.e., after the "direction" parameter has been applied), the elements before the cursor are skipped over when preparing the response. Furthermore, a result set constrained with the "limit" query parameter includes metadata values [RFC7952] called "next" and "previous", which contains cursor values to the next and previous result-sets. These next and previous cursor values are opaque index values for the underlying system's database, e.g. a key or other information needed to efficiently access the selected result-set. These "next" and "previous" metadata values work as Hypermedia as the Engine of Application State (HATEOAS) links [REST-Dissertation]. This means that the server does not keep any stateful information about the "next" and "previous" cursor or the current page. Due to their ephemeral nature, cursor values are never cached.

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, e.g. a list key or other information to efficiently access the selected result-set. 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-app-tag in the error output when this occurs.

Conformance
The "cursor" query parameter MUST be supported for all "config true" lists and SHOULD be supported for all "config false" lists. It is however optional to support the "cursor" query parameter for "config false" lists and the support must be signaled by the server per list.

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

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

3.1.7. The "limit" Query Parameter

Description
: The "limit" query parameter limits the number of entries returned from the working result-set (i.e., after the "offset" parameter has been applied). Any list or leaf-list that is limited includes, somewhere in its encoding, a metadata value [RFC7952] called "remaining", a positive integer indicating the number of elements that were not included in the result-set by the "limit" operation, or the value "unknown" in case, e.g., the server determines that counting would be prohibitively expensive.

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

Allowed Values
: The allowed values are unsigned integers. Note that passing zero (0) as limit, SHOULD quickly return the empty set. This can be useful to test if a query is valid without stressing the datastore.

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

Thusly this section introduces mechanisms whereby a server can:

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

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

> Note: The pagination performance for "config true" lists and leaf-lists is not considered as 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 [RFC9196].

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 [RFC9196]).

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
    +--ro cursor-supported?   empty
```

Comments:

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

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

4.2. Example Usage

4.2.1. Constraining a "config false" list

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

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

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

4.3. YANG Module

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

```
   <CODE BEGINS> file "ietf-list-pagination@2024-10-21.yang"

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

     import ietf-datastores {
       prefix ds;
       reference
         "RFC 8342: Network Management Datastore Architecture (NMDA)";
     }
```

```
  import ietf-yang-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
      "RFC 9196: 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) 2024 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Revised
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (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 2024-10-21 {
      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.";
    }

    md:annotation next {
      type string;
      description
        "This annotation contains the base64 encoded value of the next
         cursor in the pagination.";
    }

    md:annotation previous {
      type string;
```

```
    description
      "This annotation contains the base64 encoded value of the
       previous cursor in the pagination.";
  }

  md:annotation locale {
    type string;
    description
      "This annotation contains the locale used when sorting.

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

       For further details see references:
       RFC 5646: Tags for identifying Languages
       RFC 6365: Technology Used in Internationalization in the
                 IETF";
  }

  // Identities

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

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

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

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

```
  // Groupings

  grouping 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 locale-param-grouping {
    description
      "The grouping may be used by protocol-specific YANG modules
       to define a protocol-specific query parameter.";
    leaf locale {
      type string;
      description
        "The 'locale' parameter indicates the locale which the
         entries in the working result-set should be collated.";
    }
  }

  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-zA-z._-]*:)?[0-9a-zA-Z._-]*'
                    + '(/([0-9a-zA-Z._-]*:)?[0-9a-zA-Z._-]*)*';
          }
          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
```

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

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

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

      leaf constrained {
        type 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.";
      }
      leaf cursor-supported {
        type empty;
        description
          "Indicates that the targeted list node supports the
           'cursor' parameter.";
      }
    }
  }

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

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

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

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

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

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

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

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

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>.

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

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

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

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

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

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

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

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

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

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

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

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

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

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>.

[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>.

[REST-Dissertation] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

[I-D.ietf-netconf-list-pagination-nc] Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "NETCONF Extensions to Support List Pagination", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-nc-04, 8 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-nc-04>.

[I-D.ietf-netconf-list-pagination-rc] Watsen, K., Wu, Q., Hagsand, O., Li, H., and P. Andersson, "RESTCONF Extensions to Support List Pagination", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-rc-04, 8 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-list-pagination-rc-04>.

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 "https://example.com/ns/example-social";
     prefix es;

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

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

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

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

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

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

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

```
          }
          description
            "The member's identifier.";
        }

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

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

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

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

        container privacy-settings {
          leaf hide-network {
            type boolean;
            description
              "Hide who you follow and who follows you.";
          }
          leaf post-visibility {
            type enumeration {
              enum public {
                description
                  "Posts are public.";
              }
```

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

```
          "avatar": "BASE64VALUE=",
          "privacy-settings": {
            "post-visibility": "unlisted"
          },
          "following": ["alice", "bob"],
          "stats": {
            "joined": "2022-02-19T13:12:00Z",
            "membership-level": "standard",
            "last-activity": "2022-04-19T13:12:59Z"
          }
        }
      ]
    },
    "example-social:audit-logs": {
      "audit-log": [
        {
          "timestamp": "2020-10-11T06:47:59Z",
          "member-id": "alice",
          "source-ip": "192.168.0.92",
          "request": "POST /groups/group/2043",
          "outcome": true
        },
        {
          "timestamp": "2020-11-01T15:22:01Z",
          "member-id": "bob",
          "source-ip": "192.168.2.16",
          "request": "POST /groups/group/123",
          "outcome": false
        },
        {
          "timestamp": "2020-12-12T21:00:28Z",
          "member-id": "eric",
          "source-ip": "192.168.254.1",
          "request": "POST /groups/group/10",
          "outcome": true
        },
        {
          "timestamp": "2021-01-03T06:47:59Z",
          "member-id": "alice",
          "source-ip": "192.168.0.92",
          "request": "POST /groups/group/333",
          "outcome": true
        },
        {
          "timestamp": "2021-01-21T10:00:00Z",
          "member-id": "bob",
          "source-ip": "192.168.2.16",
          "request": "POST /groups/group/42",
```

```
          "outcome": true
        },
        {
          "timestamp": "2020-02-07T09:06:21Z",
          "member-id": "alice",
          "source-ip": "192.168.0.92",
          "request": "POST /groups/group/1202",
          "outcome": true
        },
        {
          "timestamp": "2020-02-28T02:48:11Z",
          "member-id": "bob",
          "source-ip": "192.168.2.16",
          "request": "POST /groups/group/345",
          "outcome": true
        }
      ]
    }
  }
```

A.3. Example Queries

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

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

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

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

A.3.1. The "limit" Parameter

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

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

A.3.3. The "cursor" Parameter

Noting that "cursor" 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.3.3.4. cursor="BASE64VALUE="

REQUEST

The cursor used does not exist in the datastore.

```
Target: /example-social:members/member
  Pagination Parameters:
    Where:     -
    Sort-by:   -
    Direction: -
    Offset:    -
    Limit:     -
    Cursor:    BASE64VALUE=

RESPONSE

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

A.3.4. The "direction" Parameter

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

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

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

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

A.3.4.1. direction=forwards

REQUEST

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

RESPONSE
```

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

A.3.4.2. direction=backwards

REQUEST

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

RESPONSE

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

A.3.5. The "sort-by" Parameter

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

A.3.5.1. the target node's type

The section provides three examples, one for a "leaf-list" and two for a "list", with one using a direct 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.3.6. 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

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

REQUEST

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

RESPONSE

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

A.3.6.2. match on 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",
      ...
    }
  ]
}
```

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

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

REQUEST

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

RESPONSE

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

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

A.3.7. The "locale" Parameter

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

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

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

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

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

REQUEST

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

RESPONSE

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

REQUEST

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

RESPONSE

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

REQUEST

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

RESPONSE

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

   REQUEST

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

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

   RESPONSE

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

   REQUEST

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

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

   RESPONSE

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

A.3.8. The "sublist-limit" Parameter

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

A.3.8.1. target is a list entry

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

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

This example sets the sublist-limit value '1', which returns just the first entry for all 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.3.8.2. 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.3.9. Combinations of Parameters

A.3.9.1. All six parameters at once

REQUEST

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

   RESPONSE

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

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


Acknowledgements

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

Authors' Addresses

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

Qin Wu
Huawei Technologies

Email: bill.wu@huawei.com

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

Olof Hagsand
SUNET
Email: olof@hagsand.se

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

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2024

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

1 March 2024

NETCONF Extensions to Support List Pagination
draft-ietf-netconf-list-pagination-nc-03

Abstract

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

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

Table of Contents

1. Introduction

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

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

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

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

1.1. Terminology

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

1.2. Conventions

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

2. Updates to NETCONF operations

2.1. Updates to RFC 6241

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

2.2. Updates to RFC 8526

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

3. List Pagination for NETCONF

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

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

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

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

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

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

Comments:

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

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

4. Error Reporting

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

4.1. The "offset" Query Parameter

If the "offset" query parameter value supplied is larger then the number of instances in the list or leaf-list target resource, the <rpc-error> MUST contain error-app-tag with value "offset-out-of-range".

5. YANG Module for List Pagination in NETCONF

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

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

<CODE BEGINS> file "ietf-list-pagination-nc@2024-03-01.yang"

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

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

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

  import ietf-list-pagination {
    prefix lpg;
    reference
      "RFC XXXX: List Pagination for YANG-driven Protocols";
```

```
  }

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

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

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

     Copyright (c) 2024 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Revised
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (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 2024-03-01 {
    description
      "Initial revision.";
    reference
      "RFC XXXX: NETCONF Extensions to Support List Pagination";
  }

  grouping pagination-parameters {
    description "A grouping for list pagination parameters.";
    container list-pagination {
      description "List pagination parameters.";
      uses lpg:where-param-grouping;
      uses lpg:sort-by-param-grouping;
      uses lpg:locale-param-grouping;
```

```
        uses lpg:direction-param-grouping;
        uses lpg:cursor-param-grouping;
        uses lpg:offset-param-grouping;
        uses lpg:limit-param-grouping;
        uses lpg:sublist-limit-param-grouping;
      }
    }

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

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

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

  <CODE ENDS>
```

6. IANA Considerations

6.1. The "IETF XML" Registry

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

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

6.2. The "YANG Module Names" Registry

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

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

7. Security Considerations

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

The YANG module defined in this document extends the base operations for NETCONF [RFC6241]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242].

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

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

8. References

8.1. Normative References

[I-D.ietf-netconf-list-pagination]
Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "List Pagination for YANG-driven Protocols", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-03, 1 March 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-netconf-list-pagination/>.

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

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

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

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

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

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

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

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

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

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

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

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

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

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

8.2. Informative References

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

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

Appendix A. Open Issues

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

Appendix B. Example YANG Module

The examples within this document use the "example-social" YANG module defined in Appendix A.1 of [I-D.ietf-netconf-list-pagination].

Appendix C. Example Data Set

The Example Data Set used by the examples is defined in Appendix A.2 of [I-D.ietf-netconf-list-pagination].

Appendix D. Example Queries

D.1. List pagination with all query parameters

This example mimics that Appendix A.3.9 of [I-D.ietf-netconf-list-pagination].

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

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

   Response from the NETCONF server:

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

   <lp:xml-list xmlns:lp="urn:ietf:params:xml:ns:yang:ietf-restconf-lis\
   t-pagination"
     xmlns="https://example.com/ns/example-social">
     <member lp:remaining="1">
       <member-id>eric</member-id>
       <email-address>eric@example.com</email-address>
       <password>$0$1543</password>
       <avatar>BASE64VALUE=</avatar>
       <tagline>Go to bed with dreams; wake up with a purpose.</tagline>
       <following>alice</following>
       <posts>
         <post>
           <timestamp>2020-09-17T18:02:04Z</timestamp>
           <title>Son, brother, husband, father</title>
           <body>What's your story?</body>
         </post>
       </posts>
       <favorites>
         <bits lp:remaining="2">two</bits>
       </favorites>
       <stats>
         <joined>2020-09-17T19:38:32Z</joined>
```

```
        <membership-level>pro</membership-level>
        <last-activity>2020-09-17T18:02:04Z</last-activity>
      </stats>
    </member>
    <member lp:remaining="1">
      <member-id>bob</member-id>
      <email-address>bob@example.com</email-address>
      <password>$0$1543</password>
      <avatar>BASE64VALUE=</avatar>
      <tagline>Here and now, like never before.</tagline>
      <posts>
        <post lp:remaining="2">
          <timestamp>2020-08-14T03:32:25Z</timestamp>
          <body>Just got in.</body>
        </post>
      </posts>
      <favorites>
        <decimal64-numbers lp:remaining="1">3.14159</bits>
      </favorites>
      <stats>
        <joined>2020-08-14T03:30:00Z</joined>
        <membership-level>standard</membership-level>
        <last-activity>2020-08-14T03:34:30Z</last-activity>
      </stats>
    </member>
  </lp:xml-list>
```

Acknowledgements

This work has benefited from the discussions of RESTCONF resource collection over the years, in particular, [I-D.ietf-netconf-restconf-collection] which provides enhanced filtering features for the retrieval of data nodes with the GET method and [I-D.zheng-netconf-fragmentation] which document large size data handling challenge. The authors would like to thank the following for lively discussions on list:

Andy Bierman Martin Björklund Robert Varga

Authors' Addresses

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

Qin Wu
Huawei

Email: bill.wu@huawei.com

Per Andersson
Cisco Systems
Email: perander@cisco.com

Olof Hagsand
SUNET
Email: olof@hagsand.se

Hongwei Li
HPE
Email: flycoolman@gmail.com

NETCONF Working Group
Internet-Draft
Updates: 8040 (if approved)
Intended status: Standards Track
Expires: 2 September 2024

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

1 March 2024

RESTCONF Extensions to Support List Pagination
draft-ietf-netconf-list-pagination-rc-03

Abstract

This document defines a mapping of the list pagination mechanism defined in [I-D.ietf-netconf-list-pagination] to RESTCONF [RFC8040].

This document updates RFC 8040, to declare "list" and "leaf-list" as valid resource targets for the RESTCONF GET and DELETE operations, to define GET query parameters necessary for list pagination, and to define a media-type for XML-based lists.

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 2 September 2024.

Copyright Notice

Table of Contents

1. Introduction

This document defines a mapping of the list pagination mechanism defined in [I-D.ietf-netconf-list-pagination] to RESTCONF [RFC8040].

This document updates RFC 8040, as described in Section 2.

Declaring "list" and "leaf-list" as valid resource targets for the GET operation is necessary for list pagination. Declaring these nodes as valid resource targets for the DELETE operation merely completes the solution for RESTCONF.

1.1. Terminology

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

1.2. Conventions

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

2. Updates to RFC 8040

2.1. Resource Targets

This document extends Section 3.5 of [RFC8040] to add "list" and "leaf-list" nodes (not just their entries) as valid data resources for the "GET" and "DELETE" operations.

2.2. Media Type

This document extends Section 3.2 of [RFC8040] to add a new media type, "application/yang-data+xml-list", to encode "list" and "leaf-list" nodes in XML.

The "application/yang-data+xml-list" media-type defines a pseudo top-level element called "xml-list" that is used to wrap the response set, thus ensuring that a single top-level element is returned for the XML encoding", as required by Section 4.3 of [RFC8040].

For JSON, the existing "application/yang-data+json" media type is sufficient, as the JSON format has built-in support for encoding arrays.

The "application/yang-data+xml-list" media type is registered in Section 3.2.1.

2.3. Query Parameters

This document extends Section 4.8 of [RFC8040] to add new query parameters "limit", "offset", "cursor", "direction", "sort-by", "locale", "where", and "sublist-list".

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

Name	Methods	Description
limit	GET, HEAD	Limits the number of entries returned. If not specified, the number of entries that may be returned is unbounded.
offset	GET, HEAD	Indicates the number of entries in the result set that should the skipped over when preparing the response. If not specified, then no entries in the result set are skipped.
cursor	GET, HEAD	Indicates where to start the working result set, the previous entries are skipped over. If not specified, then no entries in the result set are skipped over.
direction	GET, HEAD	Indicates the direction that the result set is to be traversed. If not specified, then the result set is traversed in the "forwards" direction.
sort-by	GET, HEAD	Indicates the node name that the result set should be sorted by. If not specified, then the result set's default order is used, per YANG's "ordered-by" statement.
locale	GET, HEAD	Specifies the locale the server should use when collating the result set. If not specified, the server chooses what locale is used for collation.
where	GET, HEAD	Specifies a filter expression that result set entries must match. If not specified, then no entries are filtered from the result set.
sublist-limit	GET, HEAD	Limits the number of entries returned returned for descendent lists and leaf-lists. If not specified, the number of entries that may be returned is unbounded.

For all of the query parameters, the query parameter is only allowed for the GET and HEAD methods on "list" and "leaf-list" data resources. A "400 Bad Request" status-line MUST be returned if used with any other method or resource type. The error-tag value "operation-not-supported" is used in this case.

Per the conformance defined in Section 3.1 of [I-D.ietf-netconf-list-pagination], all of these parameters MUST be supported for all lists and leaf-lists, but servers MAY disable the support for some or all "config false" lists, as described in Section 3.3 of [I-D.ietf-netconf-list-pagination].

2.3.1. The "limit" Query Parameter

The "limit" query parameter corresponds to the "limit" parameter defined in Section 3.1.7 of [I-D.ietf-netconf-list-pagination].

If the limit value is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.2. The "offset" Query Parameter

The "offset" query parameter corresponds to the "offset" parameter defined in Section 3.1.5 of [I-D.ietf-netconf-list-pagination].

If the offset value is invalid, a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

If the offset value exceeds the number of entries in the working result set, then a "416 Range Not Satisfiable" status-line MUST be returned with the error-type value "application", error-tag value "invalid-value", and SHOULD also include the "offset-out-of-range" identity as error-app-tag value.

2.3.3. The "cursor" Query Parameter

The "cursor" querey parameter corresponds to the "cursor" parameter defined in Section 3.1.6 of [I-D.ietf-netconf-list-pagination].

If the cursor value is unknown, i.e. the key does not exist, a "404 Not Found" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value", and SHOULD also include the "cursor-not-found" identity as error-app-tag value.

If the "cursor" query parameter is not supported on the target node, then a a "501 Not Implemented" status-line MUST be returned with error-type value "application" and error-tag value "operation-not-supported".

2.3.4. The "direction" Query Parameter

The "direction" query parameter corresponds to the "direction" parameter defined in Section 3.1.4 of [I-D.ietf-netconf-list-pagination].

If the direction value is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.5. The "sort-by" Query Parameter

The "sort-by" query parameter corresponds to the "sort-by" parameter defined in Section 3.1.2 of [I-D.ietf-netconf-list-pagination].

If the specified node identifier is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.6. The "locale" Query Parameter

The "locale" query parameter corresponds to the "locale" parameter defined in Section 3.1.3 of [I-D.ietf-netconf-list-pagination].

If the specified node identifier is invalid, i.e. the locale is unknown to the server, then a "501 Not Implemented" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value", and SHOULD also include the "locale-unavailable" identity in as the error-app-tag value.

2.3.7. The "where" Query Parameter

The "where" query parameter corresponds to the "where" parameter defined in Section 3.1.1 of [I-D.ietf-netconf-list-pagination].

If the specified XPath expression is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

2.3.8. The "sublist-limit" Query Parameter

The "sublist-limit" query parameter corresponds to the "sublist-limit" parameter defined in Section 3.2.1 of [I-D.ietf-netconf-list-pagination].

If the sumlist-limit value is invalid, then a "400 Bad Request" status-line MUST be returned with the error-type value "application" and error-tag value "invalid-value".

3. IANA Considerations

3.1. The "RESTCONF Capability URNs" Registry

This document registers six capabilities in the RESTCONF Capability URNs [RFC8040] maintained at https://www.iana.org/assignments/restconf-capability-urns/restconf-capability-urns.xhtml. Following the instructions defined in Section 11.4 of [RFC8040], the below registrations are requested:

All the registrations are to use this document (RFC XXXX) for the "Reference" value.

Index	Capability Identifier
:limit	urn:ietf:params:restconf:capability:limit:1.0
:offset	urn:ietf:params:restconf:capability:offset:1.0
:cursor	urn:ietf:params:restconf:capability:cursor:1.0
:direction	urn:ietf:params:restconf:capability:direction:1.0
:sort-by	urn:ietf:params:restconf:capability:sort-by:1.0
:locale	urn:ietf:params:restconf:capability:locale:1.0
:where	urn:ietf:params:restconf:capability:where:1.0
:sublist-limit	urn:ietf:params:restconf:capability:sublist-limit:1.0

3.2. The "Media Types" Registry

This document registers one media type in the "application" subregistry of the Media Types registry [RFC6838] [RFC4855] maintained at https://www.iana.org/assignments/media-types/media-types.xhtml#application. Following the format defined in [RFC4855], the below registration is requested:

3.2.1. Media Type "application/yang-data+xml-list"

Type name: application

Subtype name: yang-data+xml-list

Required parameters: None

Optional parameters: None

Encoding considerations: 8-bit
Each conceptual YANG data node is encoded according to the XML Encoding Rules and Canonical Format for the specific YANG data node type defined in [RFC7950].

Security considerations: Security considerations related to the generation and consumption of RESTCONF messages are discussed in Section 12 of RFC 8040. Additional security considerations are specific to the semantics of particular YANG data models. Each YANG module is expected to specify security considerations for the YANG data defined in that module.

Interoperability considerations: RFC XXXX specifies the format of conforming messages and the interpretation thereof.

Published specification: RFC XXXX

Applications that use this media type: Instance document data parsers used within a protocol or automation tool that utilize the YANG Patch data structure.

Fragment identifier considerations: Fragment identifiers for this type are not defined. All YANG data nodes are accessible as resources using the path in the request URI.

Additional information:

Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): None
Macintosh file type code(s): "TEXT"

Person & email address to contact for further information:
See the Authors' Addresses section of RFC XXXX.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See the Authors' Addresses section of RFC XXXX.

Change controller: Internet Engineering Task Force
(mailto:iesg@ietf.org).

Provisional registration? (standards tree only): no

4. Security Considerations

This document introduces protocol operations for paging through data already provided by the RESTCONF protocol, and hence does not introduce any new security considerations.

This document does not define a YANG module and hence there are no data modeling considerations beyond those discussed in [I-D.ietf-netconf-list-pagination].

5. References

5.1. Normative References

[I-D.ietf-netconf-list-pagination]
Watsen, K., Wu, Q., Andersson, P., Hagsand, O., and H. Li, "List Pagination for YANG-driven Protocols", Work in Progress, Internet-Draft, draft-ietf-netconf-list-pagination-03, 1 March 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-netconf-list-pagination/>.

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

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

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

5.2. Informative References

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

[RFC4855] Casner, S., "Media Type Registration of RTP Payload Formats", RFC 4855, DOI 10.17487/RFC4855, February 2007, <https://www.rfc-editor.org/info/rfc4855>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

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

Appendix A. Example YANG Module

The examples within this document use the "example-social" YANG module defined in Appendix A.1 of [I-D.ietf-netconf-list-pagination].

Appendix B. Example Data Set

The Example Data Set used by the examples is defined in Appendix A.2 of [I-D.ietf-netconf-list-pagination].

Appendix C. Example Queries

C.1. List pagination with all query parameters

This example mimics that Appendix A.3.9 of [I-D.ietf-netconf-list-pagination].

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

GET /restconf/ds/ietf-datastores:running/example-social:members/memb\
er?where=//stats//joined[starts-with(@timestamp,'2020')]&sort-by=tim\
estamp&direction=backwards&offset=2&limit=2&sublist-limit=1  HTTP/1.1
Host: example.com
Accept: application/yang-data+xml-list
```

Response from the RESTCONF server:

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

   HTTP/1.1 200 OK
   Date: Thu, 26 Jan 2017 20:56:30 GMT
   Server: example-server
   Last-Modified: Thu, 26 Jan 2017 20:55:30 GMT
   Content-Type: application/yang-data+xml-list

   <lp:xml-list xmlns:lp="urn:ietf:params:xml:ns:yang:ietf-restconf-lis\
   t-pagination"
     xmlns="https://example.com/ns/example-social">
     <member lp:remaining="1">
       <member-id>eric</member-id>
       <email-address>eric@example.com</email-address>
       <password>$0$1543</password>
       <avatar>BASE64VALUE=</avatar>
       <tagline>Go to bed with dreams; wake up with a purpose.</tagline>
       <following>alice</following>
       <posts>
         <post>
           <timestamp>2020-09-17T18:02:04Z</timestamp>
           <title>Son, brother, husband, father</title>
           <body>What's your story?</body>
         </post>
       </posts>
       <favorites>
         <bits lp:remaining="2">two</bits>
       </favorites>
       <stats>
         <joined>2020-09-17T19:38:32Z</joined>
         <membership-level>pro</membership-level>
         <last-activity>2020-09-17T18:02:04Z</last-activity>
       </stats>
     </member>
     <member lp:remaining="1">
       <member-id>bob</member-id>
       <email-address>bob@example.com</email-address>
       <password>$0$1543</password>
       <avatar>BASE64VALUE=</avatar>
       <tagline>Here and now, like never before.</tagline>
       <posts>
         <post lp:remaining="2">
           <timestamp>2020-08-14T03:32:25Z</timestamp>
           <body>Just got in.</body>
         </post>
       </posts>
       <favorites>
         <decimal64-numbers lp:remaining="1">3.14159</bits>
```

```
      </favorites>
      <stats>
        <joined>2020-08-14T03:30:00Z</joined>
        <membership-level>standard</membership-level>
        <last-activity>2020-08-14T03:34:30Z</last-activity>
      </stats>
    </member>
  </lp:xml-list>
```

C.2. Deletion of a leaf-list

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

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

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

Response from the RESTCONF server:

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

Acknowledgements

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

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: 2 September 2024

JG. Cumming
Nokia
R. Wills
Cisco Systems

1 March 2024

NETCONF Private Candidates
draft-ietf-netconf-privcand-02

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 2 September 2024.

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, the client that created the private candidate configuration is the only client that has access to it over RESTCONF.

The private candidate configuration contains a full copy of the running configuration when it is created (in the same way as a branch does in a source control management system and in the same way as the candidate configuration datastore as defined in [RFC6241]). 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 or RESTCONF will display the entire configuration, including all changes made to it. Performing a <commit> operation will merge the changes from the private candidate into the running configuration (the same as a merge in source code management systems). 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 clients 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.

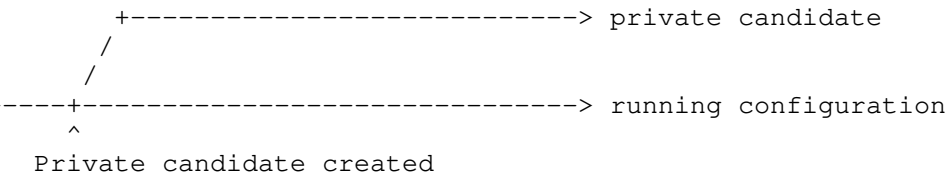
4.1. What is a private candidate

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, or the duration of the existence of the RESTCONF client. 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].

4.4.3.1. Datastore is not explicitly referenced

When the server advertises the :private-candidate capability and the client references the "{+restconf}/data" resource described in Section 3.3.1 of [RFC8040], all edits are made to the client's private candidate, and the private candidate is automatically committed.

This ensures backwards compatibility with RESTCONF clients that are not aware of private candidates, because those clients will expect their changes to be committed immediately.

4.4.3.2. Private candidate datastore is referenced in the request

When the private-candidate datastore is explicitly referenced as an NMDA datastore, edits are made to the client's private candidate, but the private candidate is not committed. To commit the changes, the client must explicitly send a commit request.

A commit request is of the form "{+restconf}/operations/ietf-netconf:commit", using the API described in Section 3.3.2 of [RFC8040]. The semantics are identical to the NETCONF <commit> operation.

Similarly, the client can perform ietf-netconf:discard-changes, ietf-netconf:validate, and ietf-netconf:cancel-commit operations (if the appropriate capabilities are implemented). The semantics are identical to NETCONF.

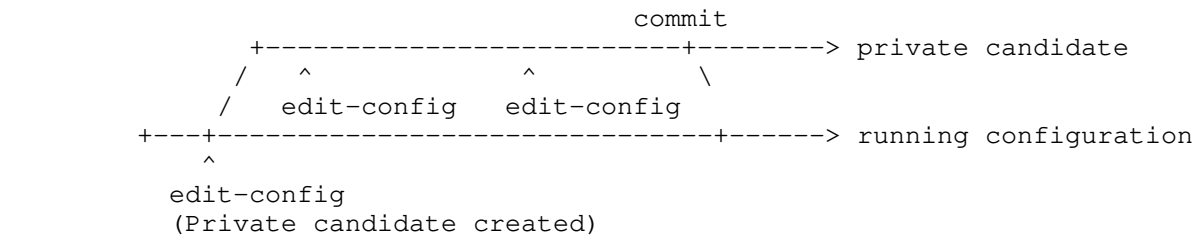
4.4.3.3. Identifying the private candidate datastore

Each RESTCONF client has its own private candidate datastore. The client (and hence the private candidate datastore) is identified using the mechanism described in Section 2.5 of [RFC8040].

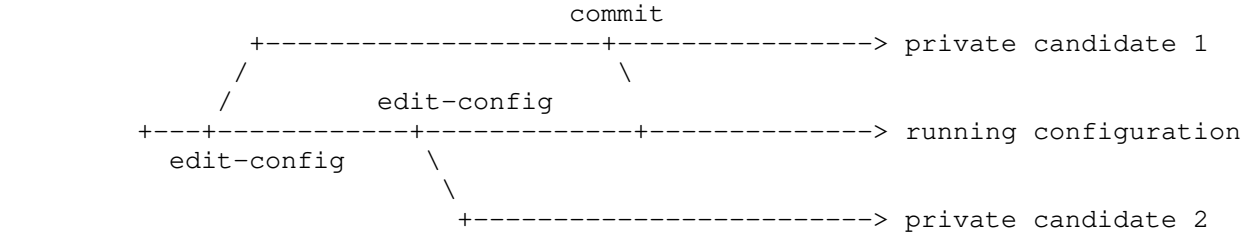
4.5. Interaction between running and private-candidate(s)

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

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



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



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

As described earlier, the client MUST be aware of changes to 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.

A good way to understand the interaction between candidates is to consider them as branches such as you might find in a source code management system.

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

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

* <commit> is issued (which MUST automatically issue an <update> operation immediately prior to committing the configuration)

* An implmentation chooses to perform an <update> operation after a change to the running configuration by any other client

It is possible for a private candidate configuration to become significantly out of sync with the running configuration should the private candidate be open for a long time, however, most NETCONF configuration activities (between the first <edit-config>/<edit-data> and a <commit>) are short-lived.

An implementation may choose, optionally, to automatically perform an <update> operation after a change to the running configuration from another client. However, this choice should be made with caution as it will replace, overwrite, or otherwise alter (depending on the servers default resolution mode, discussed later) the private candidate configuration without notifying the client

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

* Against the running configuration

4.6. Detecting and resolving conflicts

4.6.1. What is a conflict?

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

When a conflict occurs:

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

* A <commit> operation (that MUST trigger an automatic <update> operation immediately before) MUST fail. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

* A <update> operation MUST fail unless the server has explicitly configured a system-wide default resolution mode of ignore or overwrite (discussed later in this document)

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

Note: If a server implementation has chosen to automatically issue an <update> operation every time a change is made to the running configuration the server MUST have the system-wide default resolution mode set to ignore or overwrite

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.

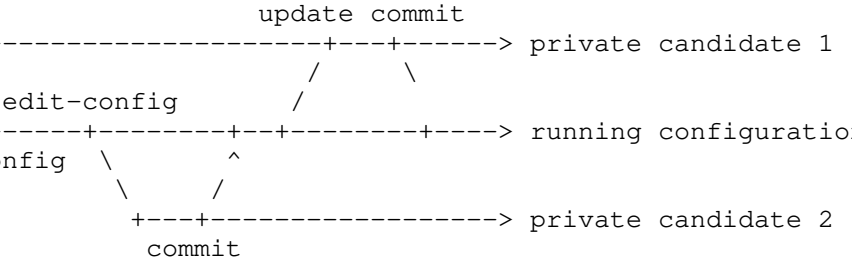
When a conflict is detected in any client triggered activity, the client MUST be informed. The client then has a number of options available to resolve the conflict.

An <update> operation uses the resolution method specified in the request, or the system default resolution mode if not specified. The <update> operation is discussed later in this document.

The following configuration data is used below to illustrate the behaviour of each resolution method:

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

The example workflow is shown in this diagram and is used for the purpose of the examples below. In these examples the reader should assume that the <update> operation is manually provided by a client working in pruvate candidate 1.



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 MUST be supported by a server.

Example:

Session 1 edits the configuration by submitting the following

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

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

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

Session 1 then sends an <update> NETCONF operation.

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

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

The resulting data in private candidate 1 is:

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

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 MUST be advertised in the :private-candidate capability by adding the default-resolution-mode parameter if the system default is anything other than revert-on-conflict. If the system default resolution mode is revert-on-conflict then advertising this in the :private-candidate capability is optional.

In this example, a server has configured a default system-wide resolution mode of overwrite which MUST be signalled with the :private-candidate capability as follows:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?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
              ?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 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. <edit-data>

The <edit-data> operation is updated to accept private-candidate as valid input to the <datastore> field. (datastore is an identityref and so the actual input will be ds:private-candidate).

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

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

4.7.2.3. <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.

4.7.2.4. <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.

4.7.2.4.1. <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.

4.7.2.5. <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).

4.7.2.6. <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).

4.7.2.7. <copy-config>

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

4.7.2.8. <delete-config>

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

4.7.2.9. <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, Kent Watsen 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

Internet Engineering Task Force
Internet-Draft
Updates: 6241, 8342, 8526, 9144 (if approved)
Intended status: Standards Track
Expires: 25 July 2025

JG. Cumming
Nokia
R. Wills
Cisco Systems

21 January 2025

NETCONF Private Candidates
draft-ietf-netconf-privcand-06

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 25 July 2025.

Copyright Notice

Table of Contents

1. Introduction

NETCONF [RFC6241] and RESTCONF [RFC8040] both provide a mechanism for one or more clients to make configuration changes to a device running as a NETCONF/RESTCONF server. Each client has the ability to make one or more configuration changes to the server's shared candidate configuration.

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

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

Many network devices support candidate configurations within the CLI interface, where a user (machine or otherwise) is able to edit a self-contained copy of a device's configuration without blocking other users from doing so.

This document specifies the extensions to the NETCONF protocol in order to support the use of private candidates. It also describes how the RESTCONF protocol can be used on a system that implements private candidates.

1.1. Requirements Language

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

1.2. Updates to RFC 6241 and RFC 8526

This document updates [RFC6241] to describe how the <edit-config>, <copy-config>, <get>, <get-config>, <commit>, <cancel-commit>, <delete-config>, <discard-changes>, <lock> and <unlock> operations work with a private candidate datastore. These updates are described in Section 4.7.2. This document also adds a new <update> operation, described in Section 4.7.1.

This document also updates [RFC8526] to describe how the <edit-data> and <get-data> operations work with a private candidate datastore. These updates are described in Section 4.7.2.

1.3. Updates to RFC 9144

This document updates [RFC9144] to augment the <compare> operation to describe how it works with the private candidate datastore. These updates are described in Section 4.7.2.7.

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, the client that created the private candidate configuration is the only client that has access to it over RESTCONF.

The private candidate configuration contains a full copy of the running configuration when it is created (in the same way as a branch does in a source control management system and in the same way as the candidate configuration datastore as defined in [RFC6241]). 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 or RESTCONF will display the entire configuration, including all changes made to it. Performing a <commit> operation will merge the changes from the private candidate into the running configuration (the same as a merge in source code management systems). This merge is performed in two steps and is described further in Section 4.7.2.1. A <discard-changes> operation will revert the private candidate to the branch's initial state or it's state at the last <update> (whichever is most recent).

All changes made to this private candidate configuration are held separately from any other candidate configuration changes, whether made by other users to the shared candidate or any other private candidate, and are not visible to or accessible by 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 behaviour may hold up valid operational activity.

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

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 clients are able to utilize private candidates to streamline network operations, particularly for machine-to-machine communication.

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

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

* Be accessible by one client only

* Be visible by one client only

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

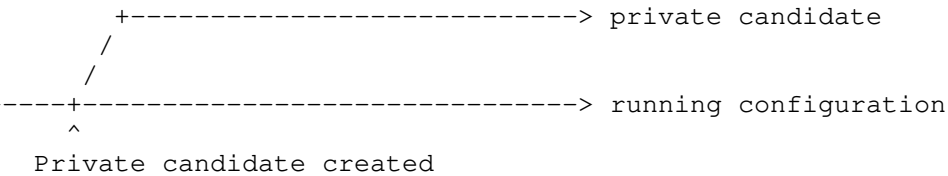
4.1. What is a private candidate

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, or the duration of the RESTCONF request. Issuing a <commit> operation will not close the private candidate but will issue an implicit <update> operation resyncing changes from the running configuration. More details on this later in this document.

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

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

As support for the shared candidate configuration datastore is a pre-requisite for the private candidate functionlity, a server MUST also advertise the :candidate capability as defined in [RFC6241].

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.

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

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

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

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

If a client sends the capability and the server does not support private candidates, the server MUST choose one of the following options:

* Ignore the capability and continue with the session without the use of private candidates (this ensures backwards compatibility with older servers).

* Close the session as the requested functionality cannot be provided.

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

All RPC requests that target or impact the shared candidate configuration datastore will implicitly target or impact the session's private candidate configuration datastore instead, and operate in exactly the same way.

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

4.4.3. RESTCONF client

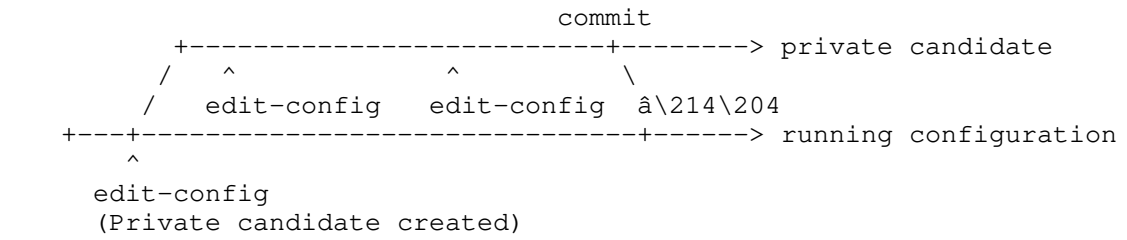
RESTCONF does not provide a mechanism for the client to advertise a capability. Therefore when a RESTCONF server advertises the :private-candidate capability, all client requests will use a private candidate when the client references the "{+restconf}/data" resource described in Section 3.3.1 of [RFC8040]. All edits are made to the client's private candidate, and the private candidate is automatically committed.

This ensures backwards compatibility with RESTCONF clients that are not aware of private candidates, because those clients will expect their changes to be committed immediately.

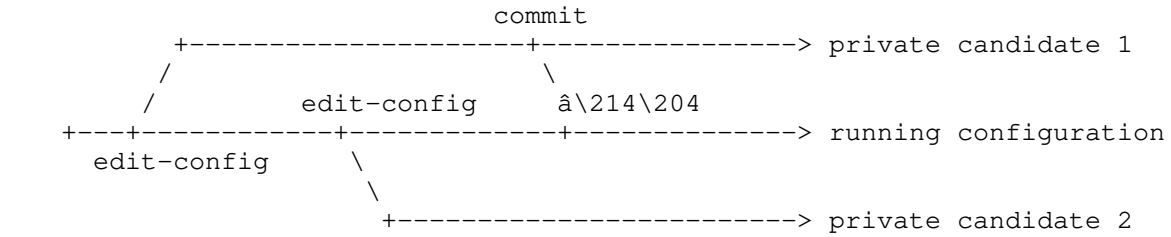
4.5. Interaction between running and private-candidate(s)

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

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



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



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

As described earlier, the client MUST be aware of changes to 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.

A good way to understand the interaction between candidates is to consider them as branches such as you might find in a source code management system.

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

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

* <commit> is issued (which MUST automatically issue an <update> operation immediately prior to committing the configuration)

* An implementation chooses to perform an <update> operation after a change to the running configuration by any other client.

It is possible for a private candidate configuration to become significantly out of sync with the running configuration should the private candidate be open for a long time, however, most NETCONF configuration activities (between the first <edit-config>/<edit-data> and a <commit>) are short-lived.

An implementation may choose, optionally, to automatically perform an <update> operation automatically on each private candidate configuration datastore after a change to the running configuration from another client. However, this choice should be made with caution as it will replace, overwrite, or otherwise alter (depending on the servers system resolution mode, discussed later) the private candidate configuration without notifying the client.

A <compare> [RFC9144] operation MAY be performed against:

* The initial creation point of the private candidate's branch

* Against the last update point of the private candidate's branch

* Against the running configuration

4.6. Detecting and resolving conflicts

4.6.1. What is a conflict?

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

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

When this happens a conflict occurs for each node modified in the running configuration that is also modified in the private candidate configuration.

A node is considered modified if:

* There is a change of any value

* There is a change of existence (or otherwise) of any list entry

* There is a change to the order of any list items in a list configured as "ordered-by user"

* There is a change of existence (or otherwise) of a presence container

* There is a change of any component member of a leaf-list

* There is a change to the order of any items in a leaf-list configured as "ordered-by user"

* There is a change of existance (or otherwise) of a leaf with an Empty type definition

* There is a change to any YANG metadata associated with the node

A server MAY choose to add additional checks over and above the above list.

If a server implements the transaction ID feature then this MAY be considered as part of detecting a conflict.

Examples of conflicts include:

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

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

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 server running an <update> operation, such as when a <commit> operation is performed by the client in the private candidate session.

When a conflict is identified that node is marked by the server as "in conflict" in the private candidate. This "in conflict" status does not propagate back up the tree to the parent node(s). Each node in the ancestral tree is evaluated as in conflict or otherwise on its own merits. The "in conflict" marker remains until the conflict is resolved on that node.

When a conflict occurs:

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

* A <commit> operation (that MUST trigger an automatic <update> operation immediately before) MUST fail irrespective of any system-wide default resolution-mode. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

* An <update> operation triggered by a client (excluding updates triggered by the <commit> operation) MUST fail unless the server has explicitly configured a system-wide resolution-mode of prefer-candidate or prefer-running (discussed later in this document).

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

Note: If a server implementation has chosen to automatically issue an <update> operation every time a change is made to the running configuration, the server will use the system-wide system resolution mode. If this resolution mode is prefer-candidate or prefer-running the conflicts will be resolved using those rules. If the resolution mode is set to revert-on-conflict the semantics are the same as the prefer-candidate method, however, all changes, whether in conflict or otherwise will be marked in the private candidate as "in-conflict". This means that any subsequent <commit> will fail until the client makes a conscious choice to resolve them.

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:

* Edit the candidate configuration nodes marked as "in conflict". When a node marked as "in conflict" is subsequently edited the "in conflict" marker on that node is removed.

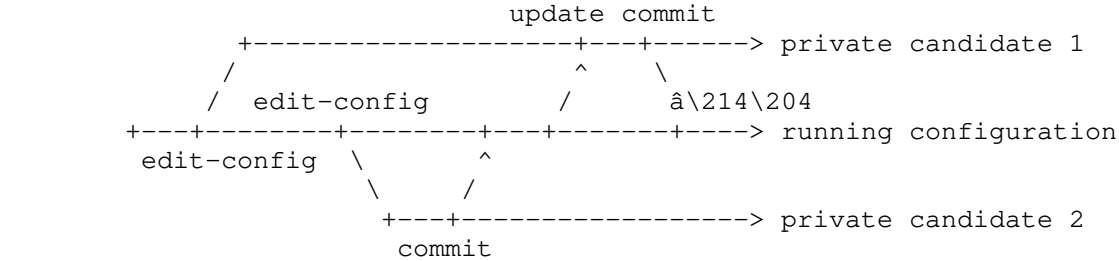
* Issue an <update> operation with a specific resolution-mode.

The <update> operation uses the resolution-mode specified in the request, or the system resolution mode in specific circumstances, to resolve the conflicts. The <update> operation is discussed later in this document.

The following configuration data is used below to illustrate the behaviour of each resolution method:

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

The example workflow is shown in this diagram and is used for the purpose of the examples below. In these examples the reader should assume that the <update> operation is manually provided by a client working in private candidate 1. The update operation triggered by a system upon commit is not shown.



There are three defined resolution methods:

4.6.3.1. Prefer-candidate

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 4.6.3.

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

Example:

Session 1 edits the configuration by submitting the following

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

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

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

Session 1 then sends an <update> NETCONF operation.

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

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

The resulting data in private candidate 1 is:

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

4.6.3.2. Prefer-running

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 4.6.3.

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

Example:

Session 1 edits the configuration by submitting the following

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

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

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

Session 1 then sends an <update> NETCONF operation.

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

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

The resulting data in private candidate 1 is:

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

4.6.3.3. Revert-on-conflict

Reminder: The starting configuration and workflow used to illustrate this resolution method is detailed in Section 4.6.3.

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

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

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

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

This resolution method MUST be supported by a server.

Example:

Session 1 edits the configuration by submitting the following:

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

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

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

Session 1 then sends an <update> NETCONF operation.

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

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

The resulting data in private candidate 1 is:

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

4.6.4. System resolution mode and advertisement of this mode

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

The system resolution mode only takes effect if a server implementation performs an automatic update to all private candidate configurations following a commit to running.

The system resolution mode MUST be advertised in the :private-candidate capability by adding the default-resolution-mode parameter if the system default is anything other than revert-on-conflict. If the system default resolution mode is revert-on-conflict then advertising this in the :private-candidate capability is optional.

In this example, a server has configured a default system-wide resolution mode of prefer-running which MUST be signalled with the :private-candidate capability as follows:

```
urn:ietf:params:netconf:capability:private-candidate:1.0
              ?default-resolution-mode=prefer-running
```

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
              ?supported-resolution-modes=revert-on-conflict,prefer-candidate
```

4.7. NETCONF operations

4.7.1. New NETCONF operations

4.7.1.1. <update>

The <update> operation triggers a rebase of the private candidate configuration against the running configuration. This rebase is conceptually described as replacing the candidate configuration with the current running configuration at that point in time and replaying the configuration changes from the candidate against it, identifing any conflicts as this process progresses.

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

The <update> operation is atomic. This means that the entire <update> operation succeeeds or the entire <update> operation MUST fail.

The <update> operation MUST be implicitly triggered by a specific NETCONF session issuing a <commit> operation when using private candidates. This implicit <update> operation always has a resolution mode of revert-on-conflict. The actual order of operations in the server MUST be to issue the implicit <update> operation first and then the <commit> operation.

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

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)
* prefer-candidate
* prefer-running

4.7.2. Updated NETCONF operations

Specific NETCONF operations altered by this document are listed in this section.

4.7.2.1. <commit>

The behaviour of the <commit> operation is updated such that the issuing a <commit> MUST follow a two step process in order to copy the proposed private configuration into the running configuration. These steps are:

1. An implicit update operation MUST be performed by the server using the resolution-mode revert-on-conflict (described later in this document).

2. On successful completion of step 1, the private candidate configuration is copied into the running configuration replacing the current contents.

4.7.2.1.1. Interactions with commit confirmed operations and private candidates

Nothing in this document alters the behaviour of the <confirmed>, <persist> or <persist-id> parameters and these MUST work when using the a private candidate configuration if the :confirmed-commit capability is advertised.

When a private candidate is committed using the <confirmed/> parameter and the commit operation disconnects the client's session, the configuration in the running configuration is immediately reverted and the proposed client changes are discarded.

When a private candidate is committed using the <confirmed/> parameter and the commit operation does not disconnect the client's session, and subsequently, the commit operation is either cancelled using the <cancel-commit> operation or the timeout expires, the running configuration is reverted and the proposed client changes are returned to the client's private candidate.

If a private candidate is committed using the <confirmed/> parameter and the <persist> parameter is provided, and the client subsequently disconnects its session for any reason whilst the timer is running, upon cancellation using the <cancel-commit> operation or on the expiry of the timer, the running configuration will be reverted, and the proposed client changes are discarded.

4.7.2.2. <get-config>

Performing a <get-config> operation with the candidate configuration datastore as the source, the configuration from the private candidate will be returned. If no private candidate configuration has been created, one will be created at this point.

4.7.2.3. <edit-config>

Performing an <edit-config> operation with the candidate configuration datastore as the target, the changes will be placed into the private candidate configuration datastore. If no private candidate configuration has been created, one will be created at this point.

4.7.2.4. <copy-config>

When performing a <copy-config> operation with the candidate configuration datastore as the source or target, the private candidate will be used. If no private candidate configuration has been created, one will be created at this point.

4.7.2.5. <get-data>

Performing a <get-data> operation with the candidate configuration datastore as the datastore, the configuration from the private candidate will be returned. If no private candidate configuration has been created, one will be created at this point.

4.7.2.6. <edit-data>

Performing an <edit-data> operation with the candidate configuration datastore as the datastore, the changes will be placed into the private candidate configuration datastore. If no private candidate configuration has been created, one will be created at this point.

4.7.2.7. <compare>

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

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

The <compare> operation is extended by this document to allow the ability to compare the private candidate (at its current point in time) with the same private candidate at an earlier point in time, or with another datastore.

4.7.2.7.1. <reference-point> parameter

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

* last-update
* creation-point

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

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

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

4.7.2.8. <lock>

The behaviour of the <lock> operation is updated such that locking the candidate configuration datastore will lock that session's private candidate configuration datastore only.

4.7.2.9. <unlock>

The behaviour of the <unlock> operation is updated such that unlocking the candidate configuration datastore will unlock that session's private candidate configuration datastore only.

4.7.2.10. <delete-config>

The behaviour of the <delete-config> operation is updated such that deleting the private candidate will destroy the private candidate for that session. A new one will subsequently be created on first access as described in Section 4.2.

4.7.2.11. <discard-changes>

The behaviour of the <discard-changes> operation is updated such that discarding the changes in a private candidate will reset it to the state it was when it was initially created, or to the state following the latest <update> operation, whichever is most recent. This state may not match the current running configuration.

To align the private candidate with the running configuration the <update> or <delete-config> operations may be used.

4.7.2.12. <get>

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

4.7.2.13. <cancel-commit>

The <cancel-commit> operation is unchanged. Any changes made to the running configuration are returned to the private candidate if it still exists. See Section 4.7.2.1.1 for further details.

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>.

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

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

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

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

7.2. Informative References

Appendix A. YANG modules

A.1. ietf-netconf-private-candidate@2024-09-12.yang

```
<CODE BEGINS> file "ietf-netconf-private-candidate@2024-09-12.yang"
module ietf-netconf-private-candidate {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:netconf:private-candidate:1.0";
    prefix pc;

    import ietf-nmda-compare {
        prefix cmp;
    }

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

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

        WG Chair: Per Andersson
                <per.ietf@ionio.se>

        Editor:   James Cumming
                <james.cumming@nokia.com>

        Editor:   Robert Wills
                <rowills@cisco.com>";
    description
        "NETCONF private candidate support.

        Copyright (c) 2024 IETF Trust and the persons identified as
        authors of the code. All rights reserved.
```

```
            Redistribution and use in source and binary forms, with or
            without modification, is permitted pursuant to, and subject
            to the license terms contained in, the Revised BSD License
            set forth in Section 4.c of the IETF Trustâ\200\231s Legal Provisions
            Relating to IETF Documents
            (https://trustee.ietf.org/license-info).

            This version of this YANG module is part of
            draft-ietf-netconf-privcand; see these documents for full
            legal notices.";

        revision 2024-09-12 {
            description
                "Introduce private candidate support";
            reference
                "draft-ietf-netconf-privcand:
                 Netconf Private Candidates";
        }

        feature private-candidate {
            description
                "NETCONF :private-candidate capability;
                If the server advertises the :private-candidate
                capability for a session, then this feature must
                also be enabled for that session.  Otherwise,
                this feature must not be enabled.";
            reference
                "draft-ietf-netconf-privcand";
        }

        rpc update {
            if-feature "private-candidate";
            description
                "Updates the private candidate from the running
                configuration.";
            reference
                "draft-ietf-netconf-privcand";
            input {
                leaf resolution-mode {
                    description
                        "Mode to resolve conflicts between running and
                        private-candidate configurations.";
                    default revert-on-conflict;
                    type enumeration {
                        enum revert-on-conflict;
                        enum prefer-candidate;
                        enum prefer-running;
                    }
```

```
                }
            }
        }

        augment /cmp:compare/cmp:input {
            leaf reference-point {
                reference "draft-ietf-netconf-privcand";
                if-feature "private-candidate";
                description
                    "When this leaf is provided and the source or
                    destination is the candidate datastore, operating
                    in private candidate mode, the comparison will
                    either occur between the last-update point of
                    the private candidate or the creation-point of
                    the private candidate.";
                default last-update;
                type enumeration {
                    enum last-update;
                    enum creation-point;
                }
            }
        }

    }
    <CODE ENDS>
```

Contributors

The authors would like to thank Andy Bierman, Jan Lindblad, Lori-Ann McGrath, Dylan Sadoun, Jason Sterne, Kent Watsen 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: 2 September 2024

J. Lindblad
Cisco Systems

1 March 2024

Transaction ID Mechanism for NETCONF
draft-ietf-netconf-transaction-id-03

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 2 September 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.

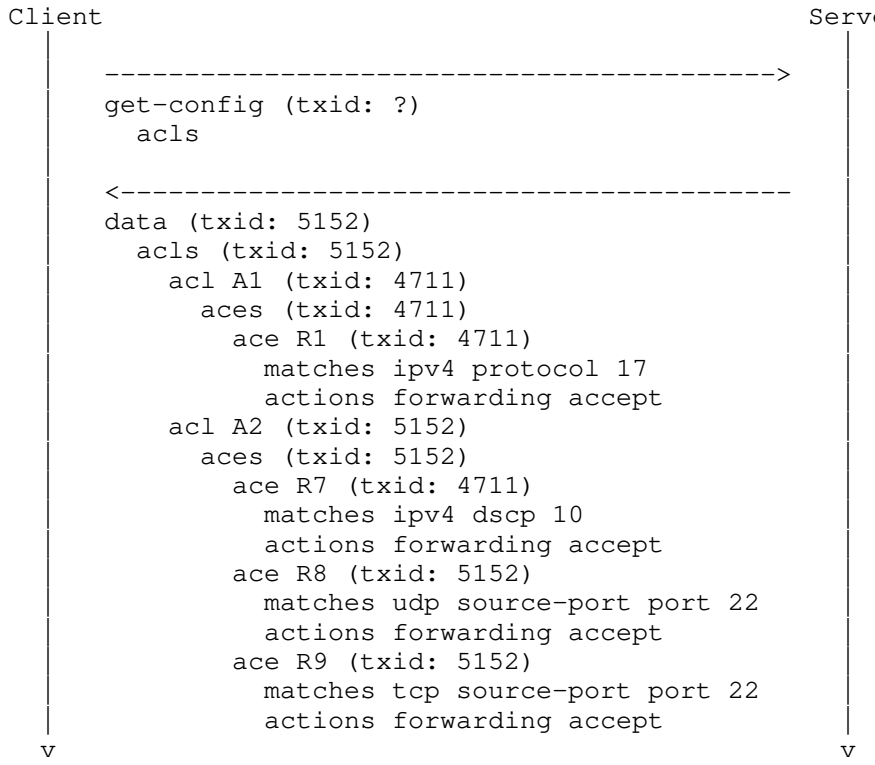


Figure 1: Initial Configuration Retrieval. The client annotated the get-config request itself with the txid request value, which makes the server return all txid values in the entire datastore, that also fall within the requested subtree filter. The most recent change seems to have been an update to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

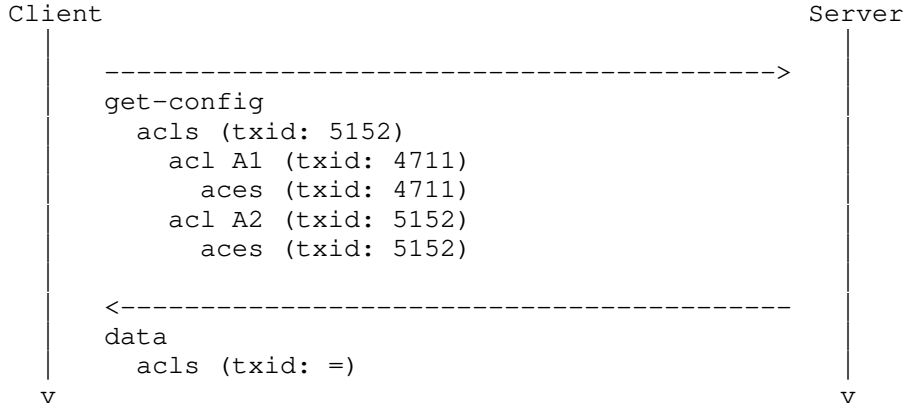


Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where the c-txid matches expectations. In this case, the server had no changes, and pruned the response at the earliest point offered by the client.

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:

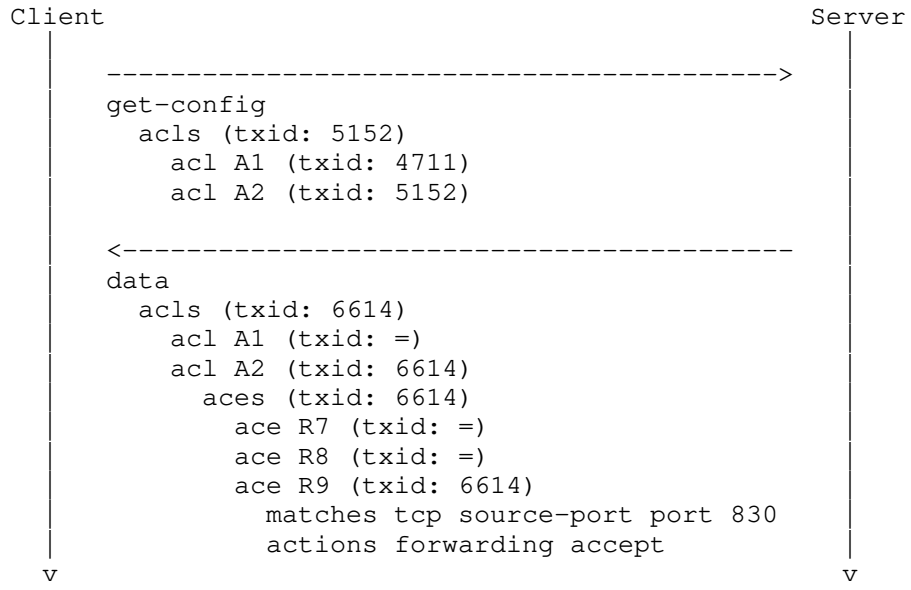


Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides updates only where changes have happened.

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.

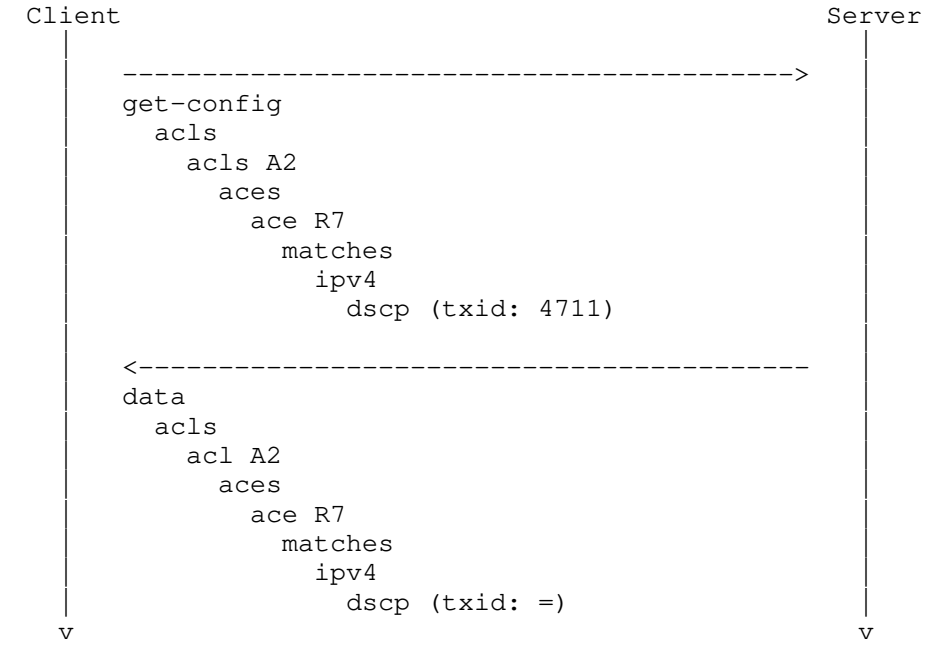


Figure 4: Versioned Nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the etag value is '=', and the leaf value is pruned.

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.

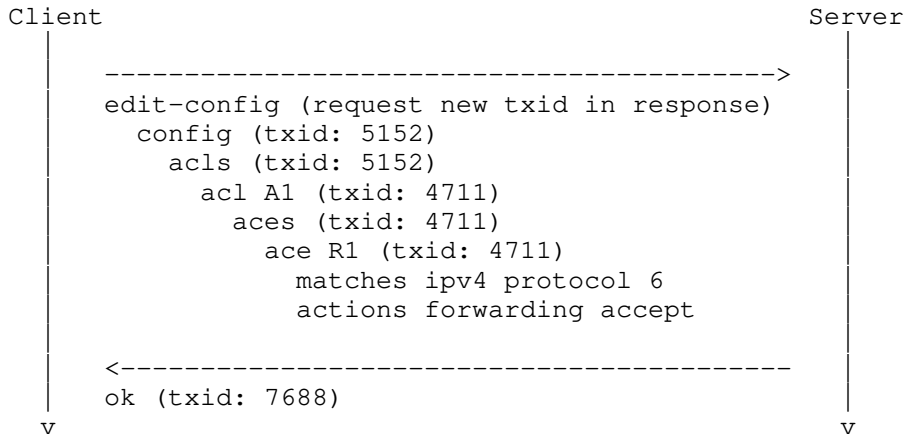


Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

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

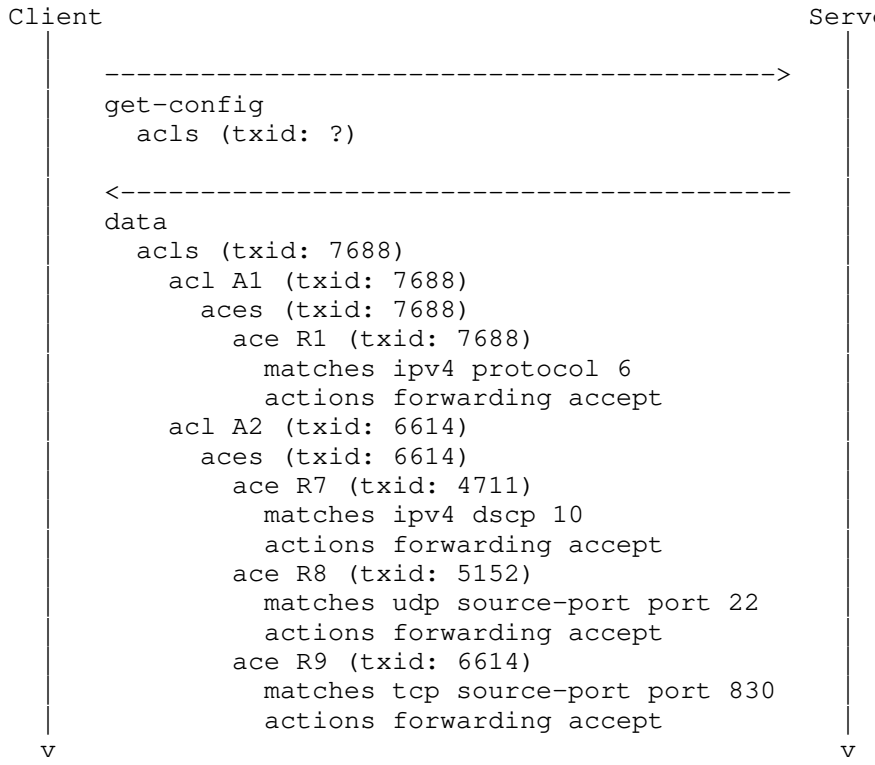


Figure 6: The txids are updated on all Versioned Nodes that were modified themselves or have a child node that was modified.

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.

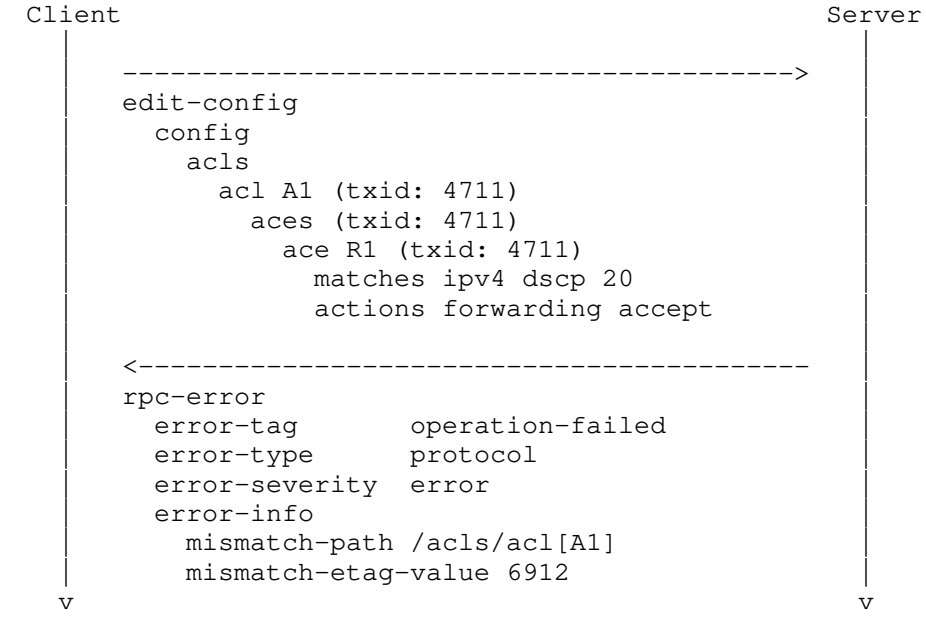


Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the c-txid it knows for this part of the configuration. Since the s-txid has changed (out of band), the server rejects the configuration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

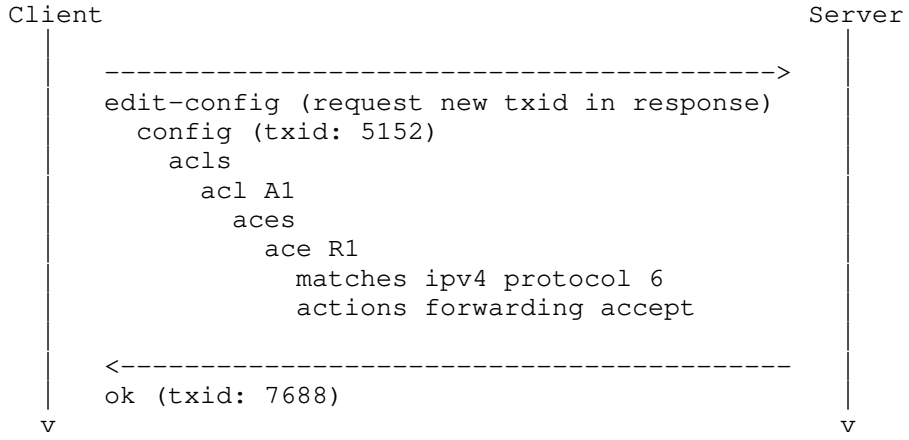


Figure 8: Conditional transaction towards the Running datastore successfully executed. As all the c-txid values specified by the client were the same or more recent in the server's Txid History, so the transaction was successfully executed.

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.

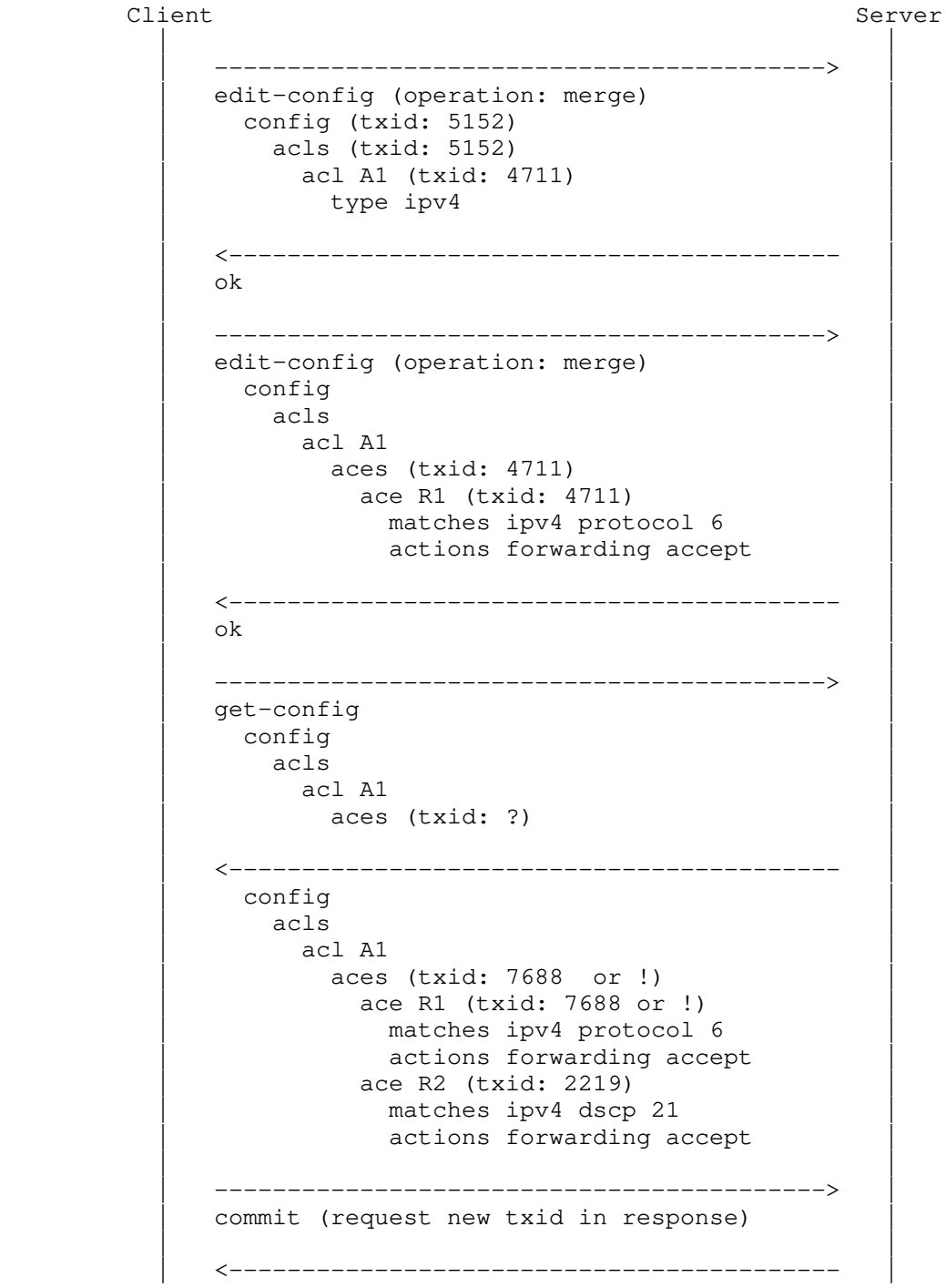

Client
Server
edit-config (operation: merge)
config (txid: 5152)
acls (txid: 5152)
acl A1 (txid: 4711)
type ipv4
ok
edit-config (operation: merge)
config
acls
acl A1
aces (txid: 4711)
ace R1 (txid: 4711)
matches ipv4 protocol 6
actions forwarding accept
ok
get-config
config
acls
acl A1
aces (txid: ?)
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:

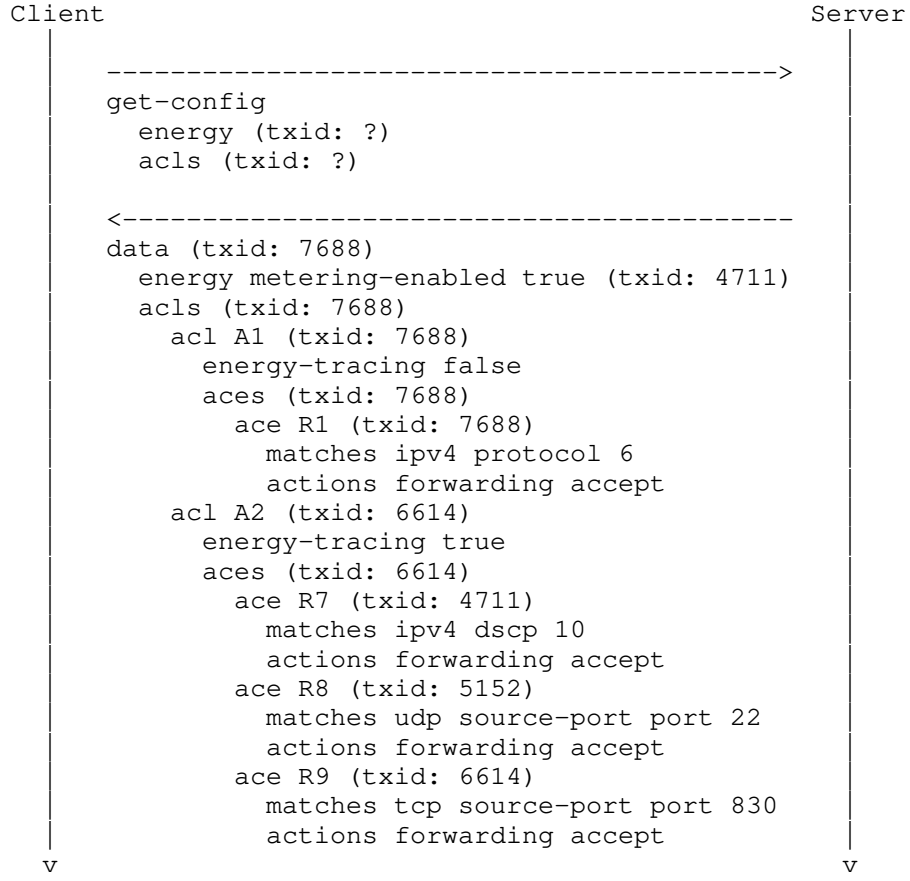


Figure 10: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

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.

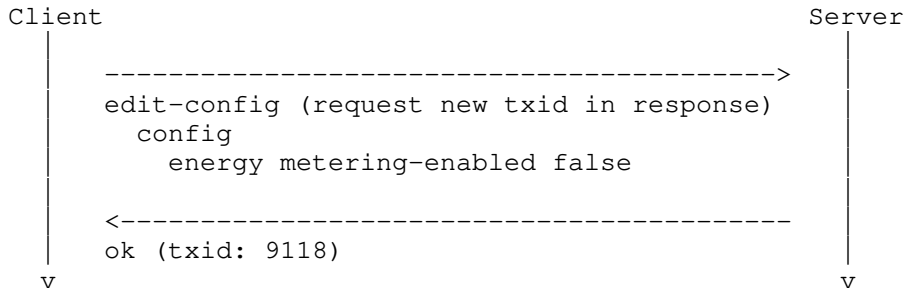


Figure 11: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in s-txid changes.

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.

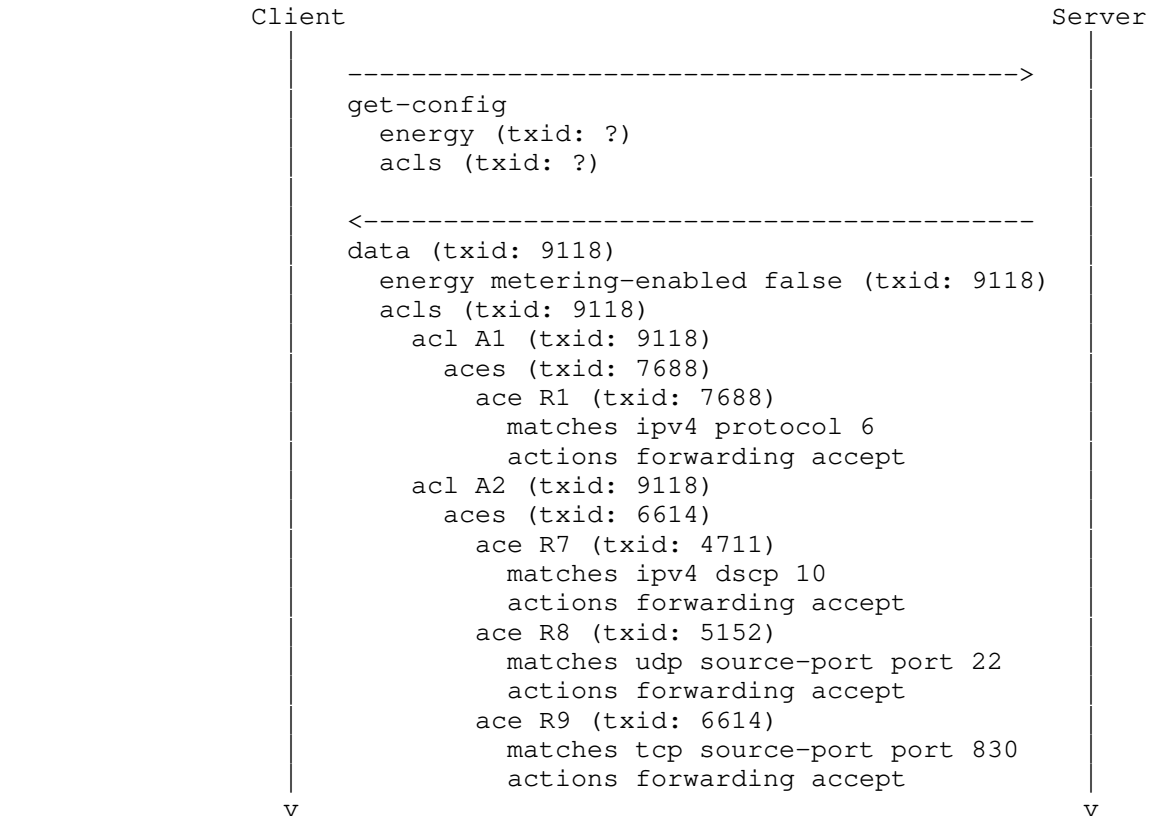


Figure 12: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when- expression. Both acl A1 and acl A2 have their txids updated, even though energy-tracing was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

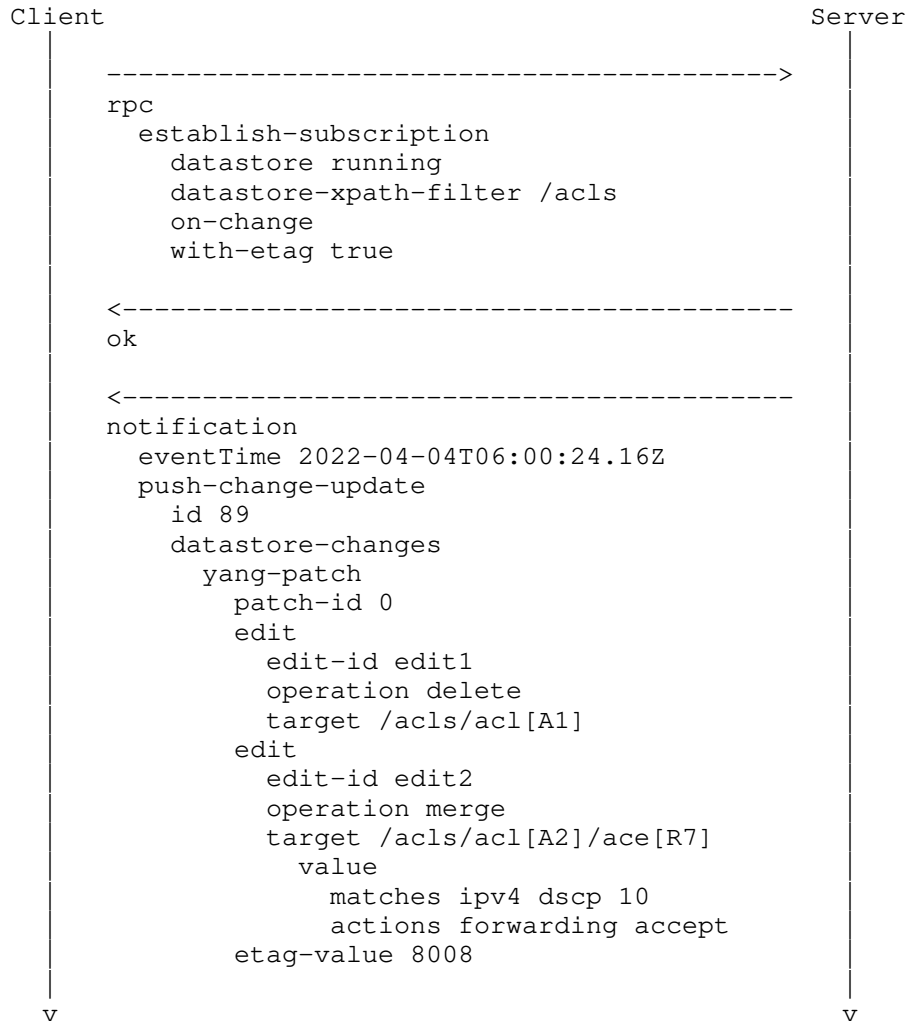


Figure 13: A client requests a YANG-Push subscription for a given path with txid value included. When the server delivers a push-change-update notification, the txid value pertaining to the entire patch is included.

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.

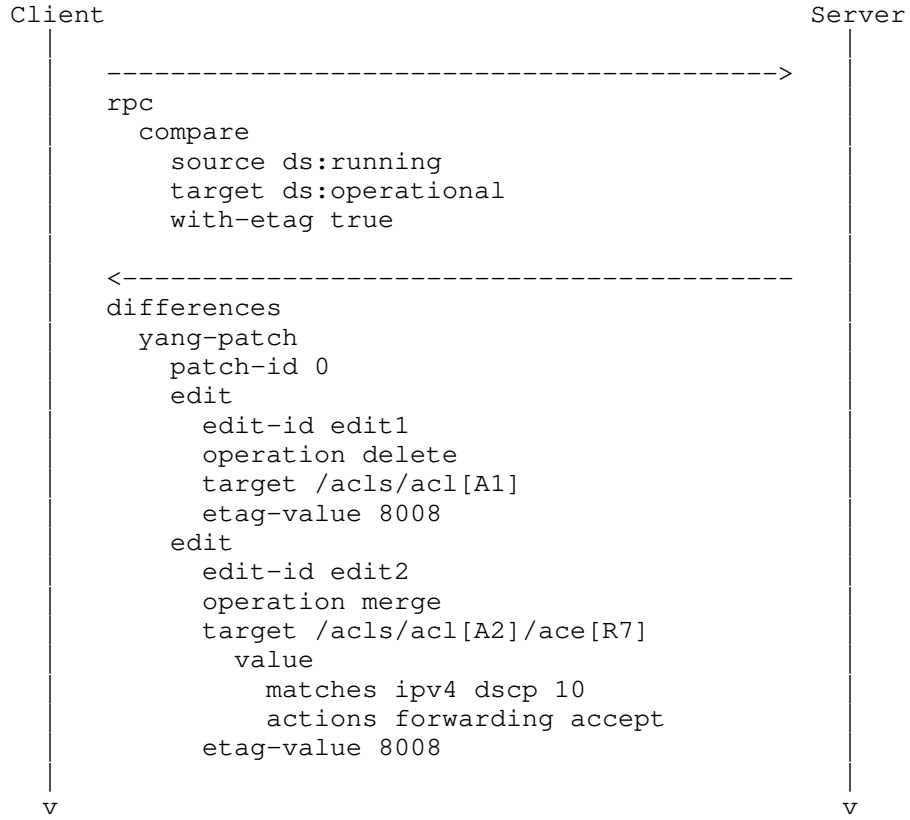


Figure 14: A client requests a NMDA Datastore compare for a given path with txid values included. When the server delivers the reply, the txid is included for each edit.

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 strings chosen freely. They MUST consist of ASCII printable characters (VCHAR), except that the etag string MUST NOT contain space, backslash or double quotes. The point of these restrictions 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. It is RECOMMENDED that the etag txid has an encoding specific suffix, especially when it is not encoded in XML. E.g. a response encoded in JSON might append "+json" at the end of the etag value. This is in line with the language in [RFC7232] and traditions in the HTTP world at large.

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-push" 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="14"
    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>
```

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>
```

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>
```

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 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 2023-03-01 {
      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 Revised BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

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

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
```

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

    revision 2022-04-01 {
      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 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 2023-05-01 {
      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 {
        description "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.";
        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 IESG.

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 -03 since -02

* Updated language slightly regarding format of etag values, and some recommendations for implementors that support etags in multiple management protocols (NETCONF, RESTCONF, ...) and encodings (XML, JSON, ...).

* Added missing normative RFC references.

* Corrected the YANG-push namespace reference.

9.2. 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".

9.3. 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.

9.4. 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

9.5. 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.

9.6. 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>.

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <https://www.rfc-editor.org/rfc/rfc8072>.

[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>.

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

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

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

[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/rfc/rfc8791>.

[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 Benoit 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: 22 April 2025

J. Lindblad
Cisco Systems

19 October 2024

Transaction ID Mechanism for NETCONF
draft-ietf-netconf-transaction-id-07

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 inefficient for synchronization. This document specifies a NETCONF extension 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 22 April 2025.

Table of Contents

1. Introduction

When a NETCONF client [RFC6241] 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 that 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 (e.g., local configuration).

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 may 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.

Note that several server implementors have built proprietary and mutually incompatible mechanisms for obtaining a transaction id from a NETCONF server. This document solves the interoperability issue.

RESTCONF, [RFC8040], defines a mechanism for detecting changes in configuration subtrees based on Entity-Tags (ETags) and Last-Modified headers. An example is depicted in Appendix B.2.2 of [RFC8040]

In conjunction with this, RESTCONF provides a way to make configuration changes conditional on the server configuration being untouched by others. This mechanism leverages conditional requests per Section 13 of [RFC9110].

This document defines similar mechanism 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, and 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]).

1.1. How to Read this Document

At the heart of this document, in chapter Txid Mechanisms (Section 4), there are two transaction-id handling mechanisms defined, the "Etag" and "Last-Modified" Transaction-id mechanisms.

The common and general principles for all transaction-id mechanisms are defined in the chapter before that, NETCONF Txid Extension (Section 3). Since the two Transaction-id mechanisms defined in this document have a lot in common, and the future might bring additional such mechanisms, this arrangement keeps the repetition to a minimum. By necessity, this chapter is a bit abstract. The details of how the principles are expressed in a specific Transaction-id mechanism follows in the Txid Mechanisms (Section 4) chapter.

Next after the central chapter with the definitions of the Transaction-id handling mechanisms, there is an extensive chapter with usage examples. This chapter is called Txid Mechanism Examples (Section 5).

Towards the end, there is also a chapter with YANG Modules (Section 6). These are necessary for a correct implementation, but reading them will not provide much for the understanding of this document. The mechanisms defined in this document are largely on the NETCONF protocol level, and most aspects cannot be described by YANG modules.

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:

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

Etag: One protocol mechanism that conforms to the definitions in the NETCONF Txid Extension (Section 3) section in this document. Also the name of the XML attribute that this mechanism uses in the NETCONF stream, and the message header used in RESTCONF.

Last-Modified: Another protocol mechanism that conforms to the definitions in the NETCONF Txid Extension (Section 3) section in this document. Also the name of the XML attribute that this mechanism uses in the NETCONF stream, and the message header used in RESTCONF.

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

Transaction-id Mechanism: A protocol implementation that fulfills the principles described in the first part, NETCONF Txid Extension (Section 3), of this document. See also Etag and Last-Modified.

Txid: Abbreviation of Transaction-id. A transaction-id is an UTF-8 string of characters. The specific format depends on the protocol mechanism used (e.g. Etag or Last-Modified).

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.

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

3. NETCONF Txid Extension

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

For servers implementing YANG-Push [RFC8641], an extension for conveying txid updates as part of subscription updates is also defined. A similar extension is also defined for servers implementing "Comparison of NMDA Datastores" [RFC9144].

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 (Section 4.1) and the 'last-modified txid' mechanism (Section 4.2). However, additional txid mechanisms may be defined in the future. Such mechanisms have to adhere to the principles defined in Section 3.2.

This document is divided into a two main 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. Sample Use Cases

The common use cases for txid mechanisms are briefly discussed in this section.

Initial configuration retrieval: When a 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 retrieve again (parts of) the server's configuration, it may be interested to leverage the txid metadata 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 metadata that 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 metadata 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 metadata 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 updated txid metadata for the changed data trees.

3.2. General Txid Principles

All servers implementing a txid mechanism MUST maintain a top level server side txid (s-txid) metadata value for each configuration datastore supported by the server. Txid mechanism implementations MAY also maintain txid metadata 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 implementations 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 a 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 (a.k.a. "license") installations or removals.

The server returning txid values for the Versioned Nodes MUST ensure that 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.

A 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 (Section 3.1.1 of [RFC8526]) containing requests for txid values, and assuming no authorization or validation error is encountered, 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.

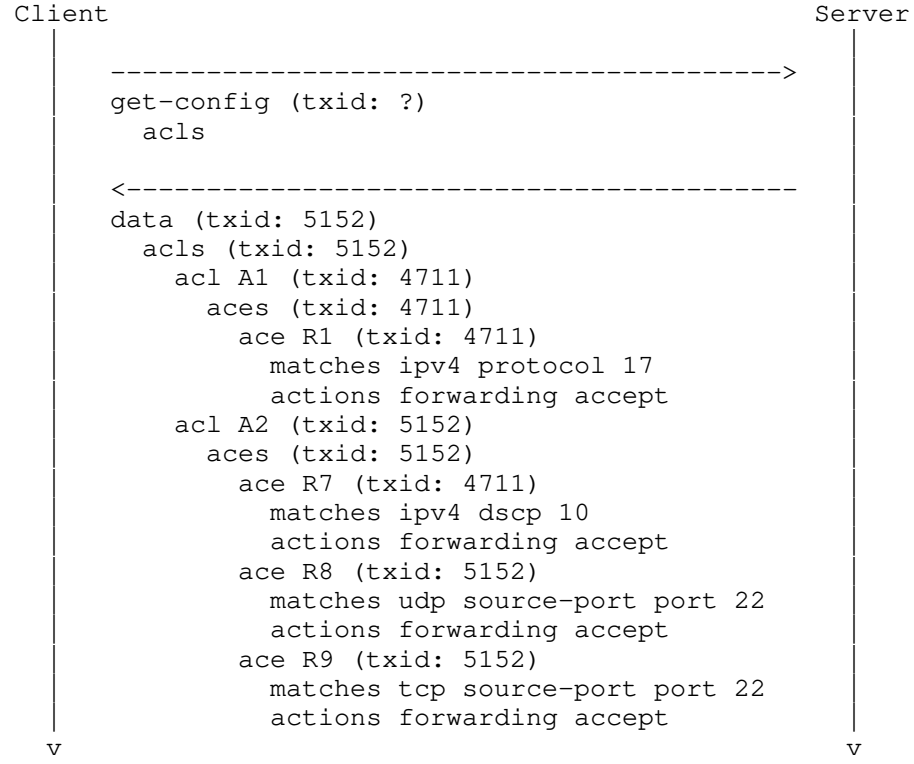


Figure 1: Initial Configuration Retrieval. The client annotated the get-config request itself with the txid request value, which makes the server return all txid values in the entire datastore, that also fall within the requested subtree filter. The most recent change seems to have been an update to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

The call flow examples in this document use a 4-digit, strictly increasing integer as txid. The same txid value is also used for all changed nodes in a given transaction. These conventions of the examples are convenient and enhances readability of the examples, but do not necessarily reflect a typical implementation.

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. Servers may expose a configuration parameter to control the history depth. Such control depends on the local server capabilities. Refer to Section 3.6.2 for more considerations about history size.

Some server implementors may decide to use a strictly 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.

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 refered to as c-txid.

When a client sends 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 (i.e., does not return) that subtree from the response. Since the client already knows the txid for that part of the data tree, or a txid that occurred more recently, it is obviously already up to date with that part of the configuration. Sending it again would be a waste of time and energy.

Table 1 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, for the purposes of these rules, temporarily 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	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.

	s-txid value for this node.	
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 any s-txid value in 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 current node according to the normal NETCONF specifications. If the current node is a Versioned Node, it MUST 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.

3.4.1. When there is No Change

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:

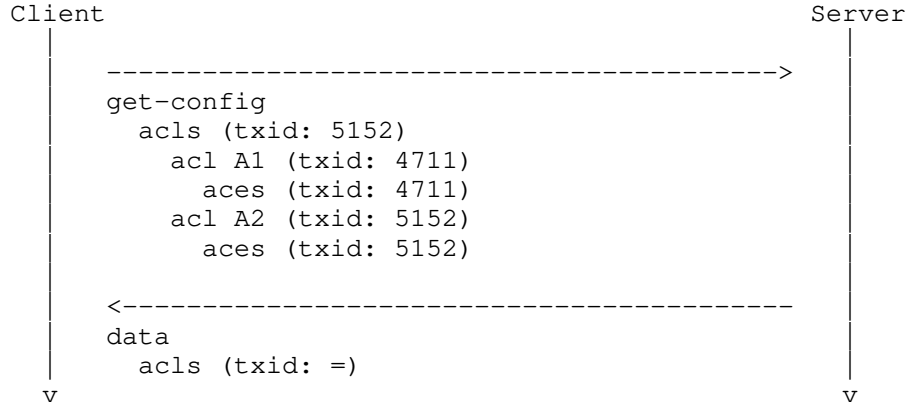


Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where the c-txid matches expectations. In this case, the server had no changes, and pruned the response at the earliest point offered by the client.

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.

3.4.2. When there is an Out-Of-Band (OOB) Change

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:

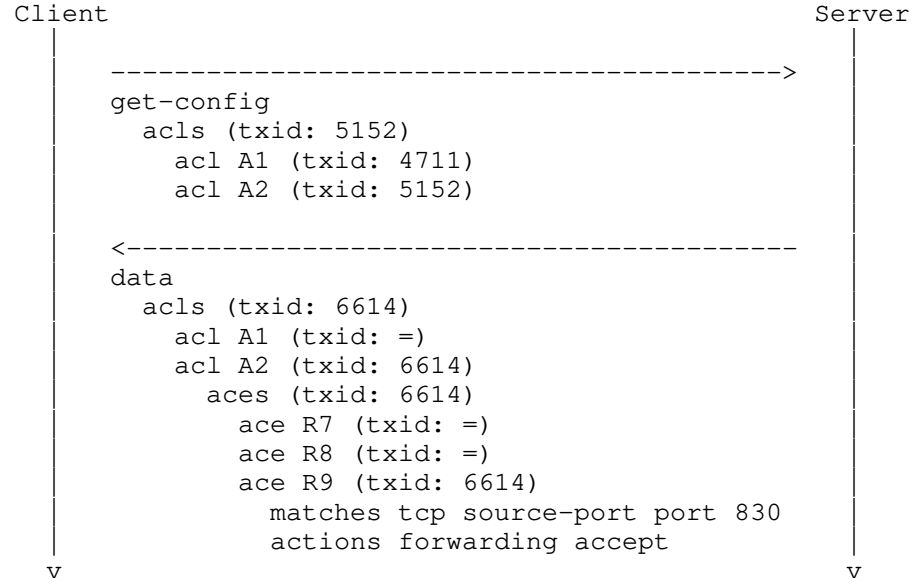


Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides updates only where changes have happened. (Txid 7770 does not appear in this subtree, so that transaction must relate to some changes elsewhere.)

In the example depicted in Figure 3, 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).

3.4.3. When a Txid value is Inherited from an Ancestor Node

In the example shown in Figure 4, the client specifies the c-txid for a node that the server does not maintain a s-txid for, i.e., it is not a Versioned Node.

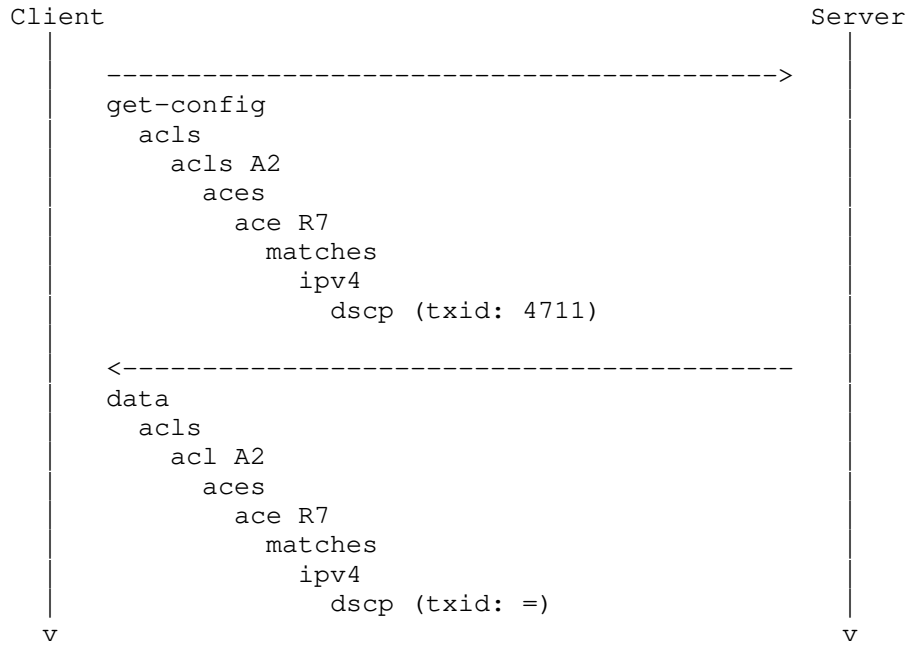


Figure 4: Versioned Nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the txid value is '=', and the leaf value is pruned.

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 use the special txid values, txid-match, txid-request, txid-unknown (e.g., "=", "?", or "!") 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 candidate.

* 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 again, i.e. the server is not required to stick with the s-txid value just returned.

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>, for example), 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>, for example) and write (<edit-config>, for example) operation.

Clients that are also interested to know the s-txid assigned to the root Versioned Node in the model immediately in the response could set a flag in the <rpc> element to request the server to return the new s-txid with the <ok> element.

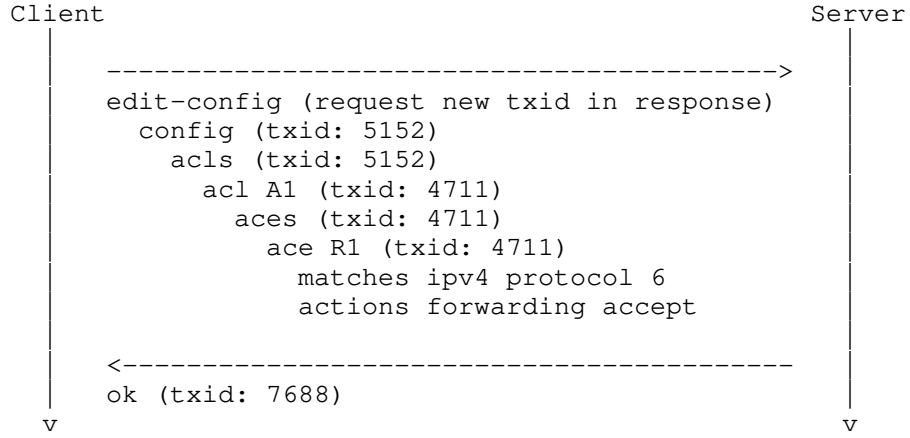


Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

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

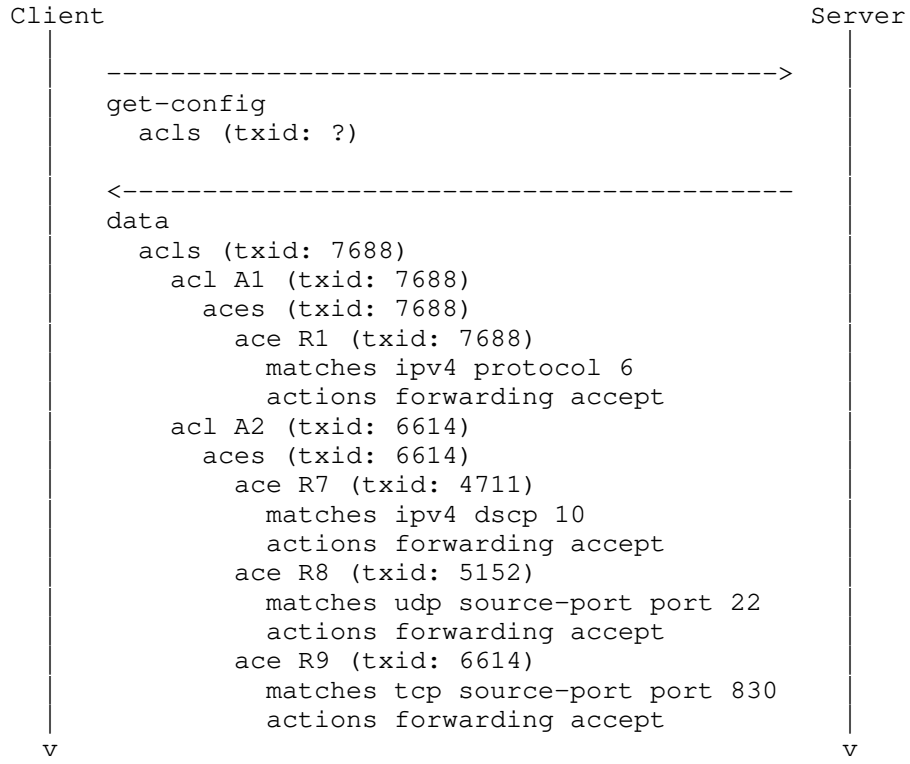


Figure 6: The txids are updated on all Versioned Nodes that were modified themselves or have a child node that was modified.

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 Changes

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 [RFC8791] containing relevant details about one of the mismatching txids. A server MAY send multiple rpc-errors when multiple txid mismatches are detected.

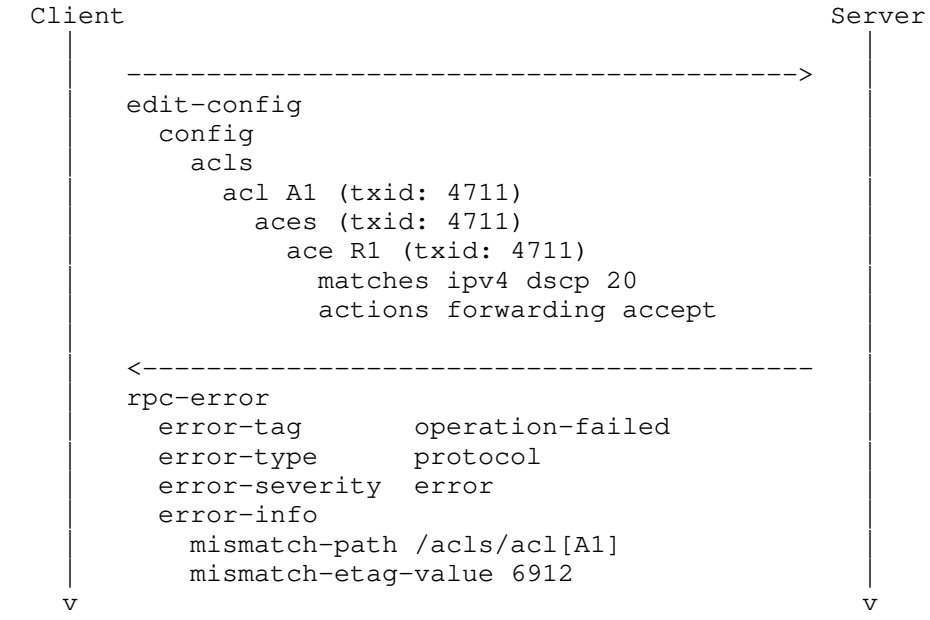


Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the c-txid it knows for this part of the configuration. Since the s-txid has changed (out of band), the server rejects the configuration 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 a single c-txid value for the tree being edited. In many 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 in the subtree below the c-txid provided.

Here the client is sending the same change as in the example above (Figure 5), but with only a single c-txid value that reflects the latest txid the client is aware of anywhere in the configuration.

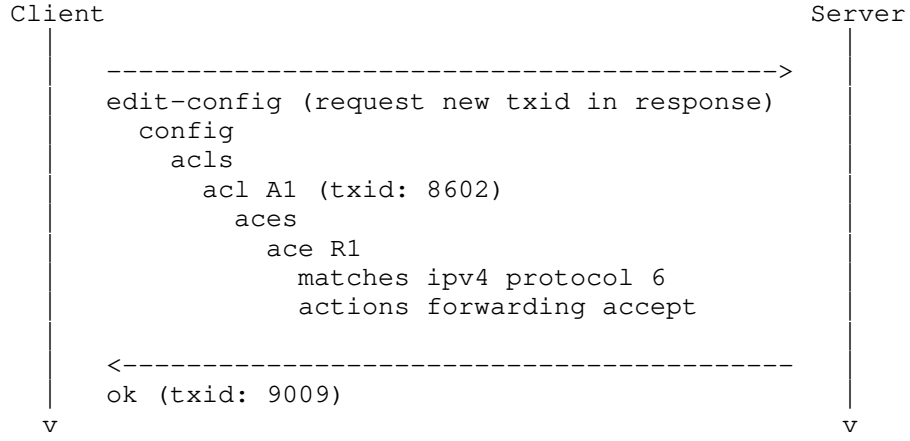


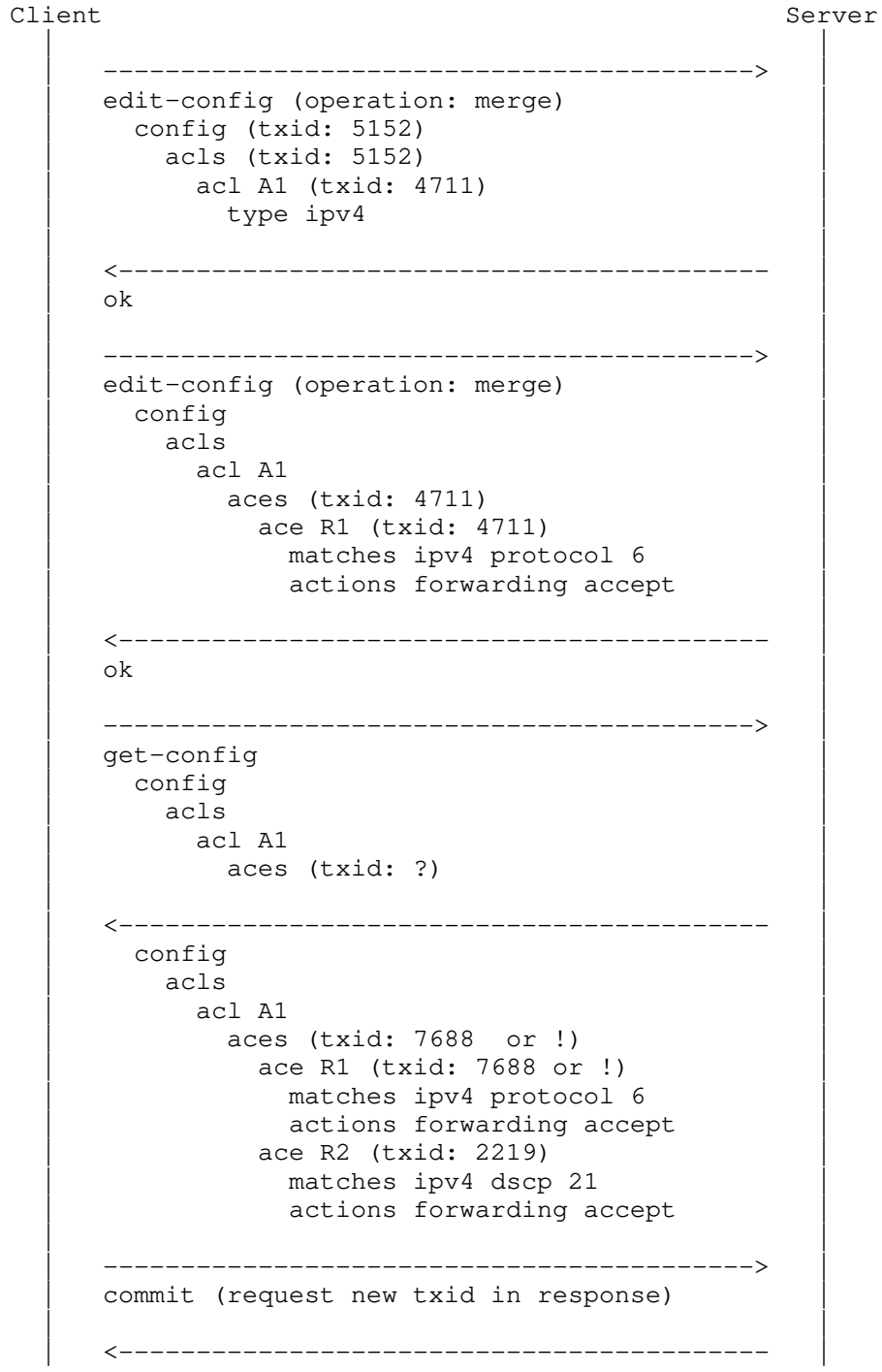
Figure 8: Conditional transaction towards the Running datastore successfully executed. As all the c-txid values specified by the client were the same or more recent in the server's Txid History, so the transaction was successfully executed.

This approach works well in the example above because the c-txid value 8602 is inherited down in the child nodes, from acl A1 to aces, ace R1, and onwards. The server compares the c-txid value 8602 with the s-txid value in the data tree. The server finds that the values do not match (e.g., s-txid 7688 for ace R1 is not equal to c-txid 8602), but finds that 8602 is a more recent txid than 7688 by looking in the server's Txid History, and therefore accepts the transaction.

Clients relying on the server's Txid History being long enough, could see their changes rejected if some of the s-txid have fallen out of the server's Txid History (e.g., if the txid 7688 happened so long ago that the it is no longer in the server's Txid History). Some servers may have a Txid History size of zero. A client specifying a single c-txid value for a change like the one above towards such a server would not be able to get the transaction accepted.

3.7. Candidate Datastore Transactions

When using 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)

```
          |   ok (txid: 7688)                          |
          v                                            v
```

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:

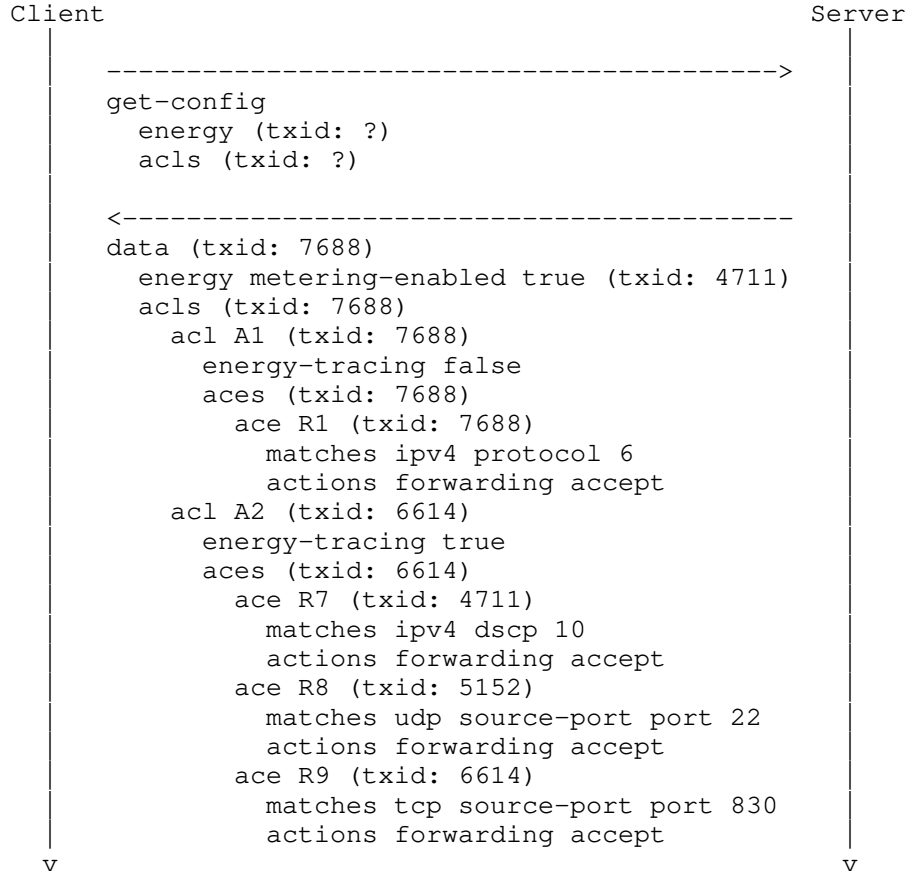


Figure 10: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

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.

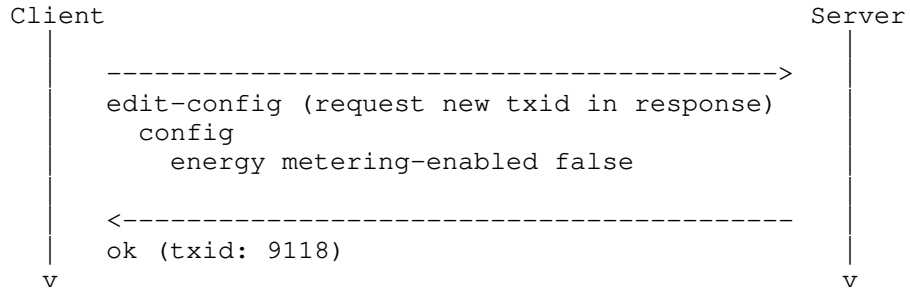


Figure 11: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in s-txid changes.

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.

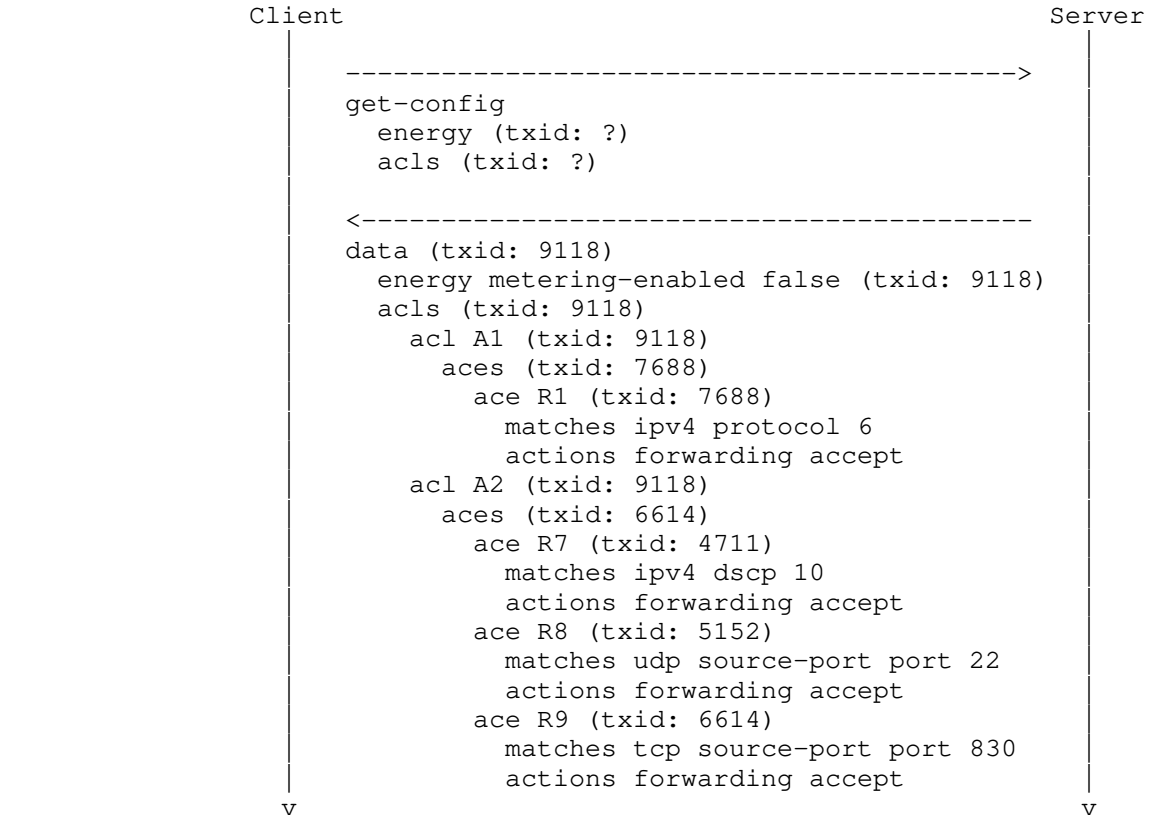


Figure 12: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when- expression. Both acl A1 and acl A2 have their txids updated, even though energy-tracing was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

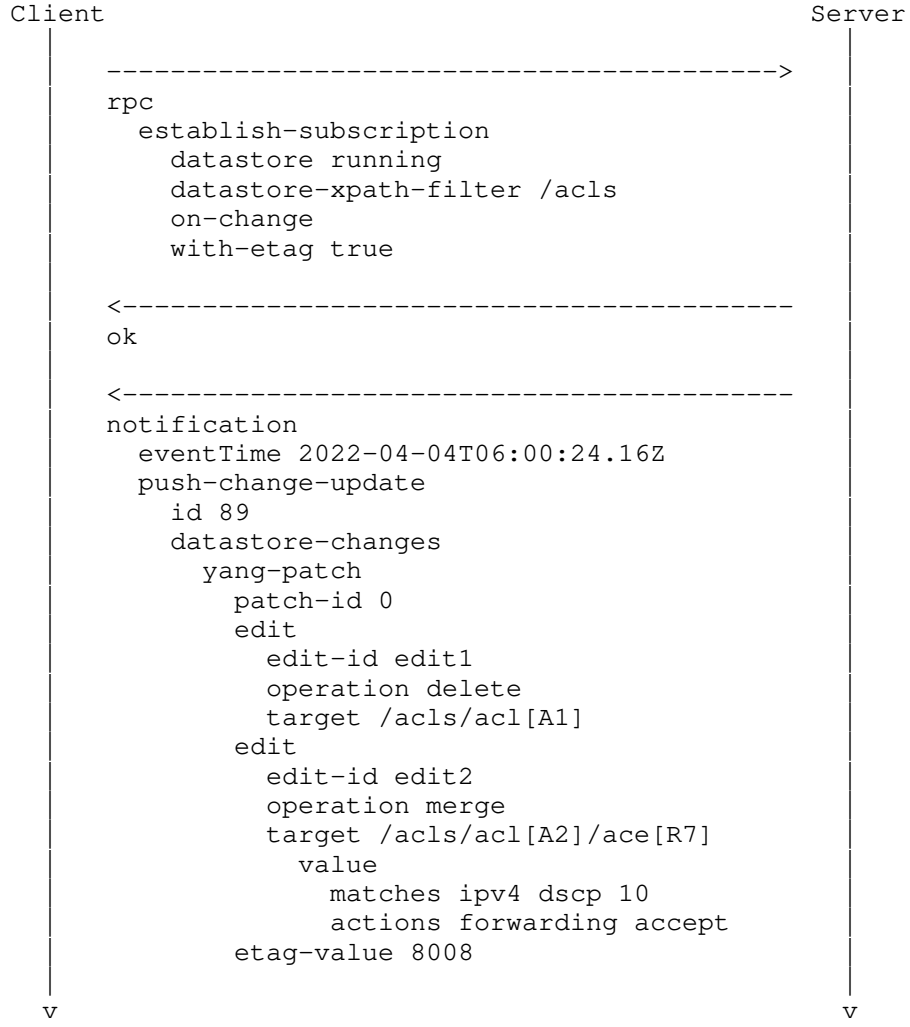


Figure 13: A client requests a YANG-Push subscription for a given path with txid value included. When the server delivers a push-change-update notification, the txid value pertaining to the entire patch is included.

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.

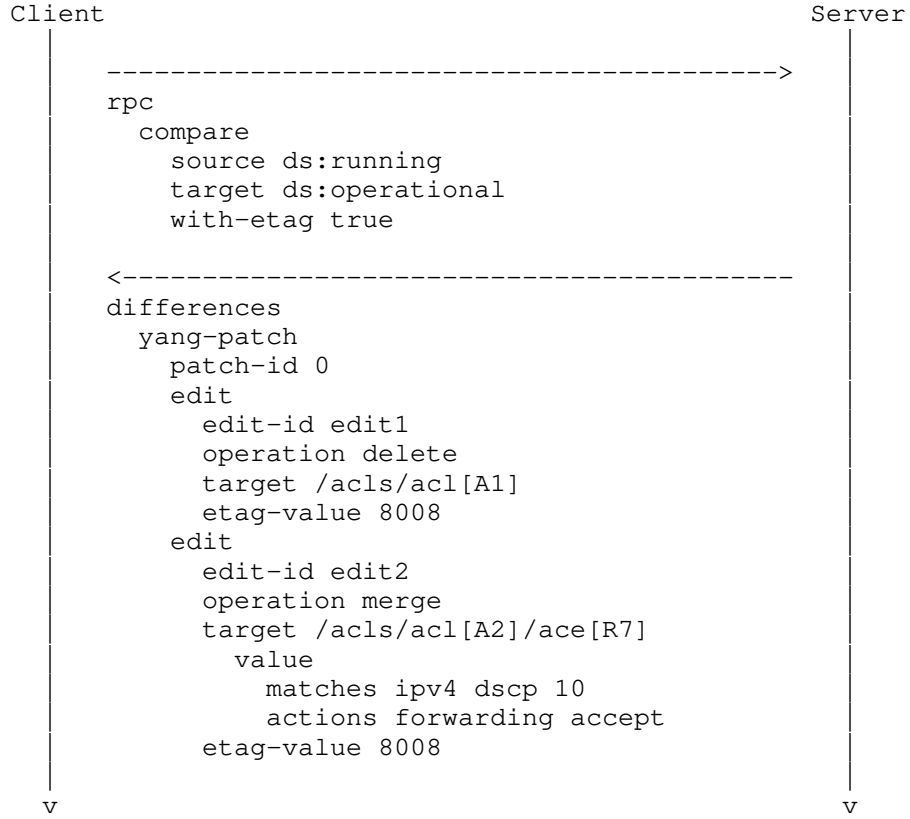


Figure 14: A client requests a NMDA Datastore compare for a given path with txid values included. When the server delivers the reply, the txid is included for each edit.

4. Txid Mechanisms

This document defines two txid mechanisms:

* The etag attribute txid mechanism (Section 4.1)

* The last-modified attribute txid mechanism (Section 4.2)

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 particular 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 strings chosen freely. They MUST consist of ASCII printable characters (VCHAR), except that the etag string MUST NOT contain space, backslash or double quotes. The point of these restrictions 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. It is RECOMMENDED that the etag txid has an encoding specific suffix, especially when it is not encoded in XML. E.g. a response encoded in JSON might append "+json" at the end of the etag value. This is in line with the language in [RFC7232] and traditions in the HTTP world at large.

The detailed rules for when to update the etag value are described in Section 3.2. These rules are chosen to be consistent with the ETag mechanism in RESTCONF, specifically Sections 3.4.1.2, 3.4.1.3 and 3.5.2 of [RFC8040].

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. Servers MUST ensure the timestamps provided are strictly increasing for as long as the server's operation is maintained.

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 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" [RFC4741] [RFC6241], xmlns:ncds="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda" [RFC8526], xmlns:sn="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications" [RFC8639], xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push" [RFC8641] [RFC8072]:

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/yp:yang-patch/ yp:edit

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

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

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

5. Txid Mechanism Examples

5.1. Initial Configuration Response

5.1.1. With etag

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="?">
    <source>
      <running/>
    </source>
  </get-config>
</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="fd6a52d9-5152-811c-a117-b99d3b723c93">
    <acls xmlns=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
          txid:etag="fd6a52d9-5152-811c-a117-b99d3b723c93">
      <acl txid:etag="2c4b50e4-4711-49f8-a2b2-2e20aebe120f">
        <name>A1</name>
        <aces txid:etag="2c4b50e4-4711-49f8-a2b2-2e20aebe120f">
          <ace txid:etag="2c4b50e4-4711-49f8-a2b2-2e20aebe120f">
            <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>
  ...
```

It is up to the server implementor to decide on the format of the etag txid value. In the example above, the server used "random" UUID values. This is one valid implementation choice.

For the etag txid examples below, we have chosen to use an etag txid value consisting of "nc" (or "cli" in some cases) followed by a strictly increasing integer. This is another valid implementation choice. This format is convenient for the reader trying to make sense of the examples, but is not an implementation requirement.

Clients have to be prepared to receive etag txid values in different formats.

Repeating the example above, but now with a server returning more human readable etag txid values, the server’s reply might 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="14"
    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>
```

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>
```

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>
```

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> file "ietf-netconf-txid@2023-03-01.yang"
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:   <https://datatracker.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 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 2023-03-01 {
    description
      "Initial revision";
    reference
      "RFC XXXX: Transaction ID Mechanism for NETCONF";
  }

  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:

        '?' This txid value is used by clients and is
            guaranteed not to match any txid on the server.

        '!' This txid value used by servers to indicate
            the node in the candidate datastore has changed
            relative to the running datastore, but not yet received
            a new txid value on the server.

        '=' This txid value used by servers to indicate
            that contents has been pruned due to txid match
            between client and server.
        ";
    }

    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 to 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> file "ietf-netconf-txid-yang-push@2022-04-01.yang"
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-netconf-txid {
    prefix ietf-netconf-txid;
    reference
      "RFC XXXX: Transaction ID Mechanism for NETCONF";
  }

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

  contact
    "WG Web:   <https://datatracker.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 Revised BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
```

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

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

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

    revision 2022-04-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Transaction ID Mechanism for NETCONF";
    }

    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> file "ietf-netconf-txid-nmda-compare@2023-05-01.yang"
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: Transaction ID Mechanism for NETCONF";
  }

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

  contact
    "WG Web:   <https://datatracker.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 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 2023-05-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Transaction ID Mechanism for NETCONF";
    }

    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 {
        description "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.";
        uses ietf-netconf-txid:txid-value-grouping;
      }
    }
  }
  <CODE ENDS>
```

7. Security Considerations

The YANG modules specified in this document define YANG types, groupings, structures and additional RPC parameters for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

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

In the YANG modules published with this document, there is no configuration, state data, new RPCs or notifications. This document defines additional XML attributes and headers, however, that merit consideration from a security perspective.

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

8.1. NETCONF Capability URN

This document requets IANA to register the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry:

Capability: :txid
Capability Identifier: urn:ietf:params:netconf:capability:txid:1.0
Reference: RFC XXXX

8.2. IETF XML Registry

This document request IANA to register four XML namespace URIs in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:netconf:txid:1.0
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare
Registrant Contact: The IESG.
XML: N/A, the requested URIs are XML namespaces.

8.3. YANG Module Names

This document requests IANA to register three module names in the "YANG Module Names" subregistry [RFC6020] within the "YANG Parameters" registry.

```
name: ietf-netconf-txid
prefix: ietf-netconf-txid
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid
maintained by IANA? N
RFC: XXXX

name: ietf-netconf-txid-yp
prefix: ietf-netconf-txid-yp
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-txid-yang-push
maintained by IANA? N
RFC: XXXX

name: ietf-netconf-txid-nmda-compare
prefix: ietf-netconf-txid-nmda-compare
namespace:
  urn:ietf:params:xml:ns:yang:ietf-netconf-txid-nmda-compare
maintained by IANA? N
RFC: XXXX
```

9. Changes

9.1. Major changes in -07 since -06

* Changed "monotonically increasing" to "strictly increasing" in multiple locations. Removed recommendation about timestamps in the last-modified txid mechanism being similar to wall clock time.

* Removed two clumsily formulated sentences stating that clients MUST NOT infer temporal order from txid values. The remaining wording states that some servers use sequences of txid values that may appear random to outside observers.

* Added brief explanation that entitlements are sometimes also known as "licenses".

* Added introductory section on "How to Read this Document"

* Added an example to highlight that the etag txid values can have different formats, and do not need to consist of strictly increasing integers, as in most of the examples.

* Changed WG URLs in YANG modules to new datatracker format, e.g. https://datatracker.ietf.org/wg/netconf/

9.2. Major changes in -06 since -05

* Many language, style, spelling and formatting improvements thanks to reviews by Reshad Rahman and Med Boucadair

* Clarified Txid History Size Consideration example

9.3. Major changes in -05 since -04

* Corrected namespace for reference to elements in ietf-yang-push

9.4. Major changes in -04 since -03

* Updated security considerations.

* Added several normative RFC references.

9.5. Major changes in -03 since -02

* Updated language slightly regarding format of etag values, and some recommendations for implementors that support etags in multiple management protocols (NETCONF, RESTCONF, ...) and encodings (XML, JSON, ...).

* Added missing normative RFC references.

* Corrected the YANG-push namespace reference.

9.6. 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".

9.7. 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.

9.8. 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

9.9. 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.

9.10. 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>.

[RFC4741] Enns, R., Ed., "NETCONF Configuration Protocol", RFC 4741, DOI 10.17487/RFC4741, December 2006, <https://www.rfc-editor.org/rfc/rfc4741>.

[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>.

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

[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>.

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <https://www.rfc-editor.org/rfc/rfc8072>.

[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>.

[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>.

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

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

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

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

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

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

[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>.

Acknowledgments

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

Author's Address

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 30 August 2024

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

27 February 2024

YANG Groupings for UDP Clients and UDP Servers
draft-ietf-netconf-udp-client-server-01

Abstract

This document defines two YANG 1.1 modules to support the configuration of UDP clients and UDP servers.

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 30 August 2024.

Copyright Notice

Table of Contents

1. Introduction

This document defines two YANG 1.1 [RFC7950] modules to support the configuration of UDP clients and UDP servers [RFC768], either as standalone or in conjunction with configuration of other layers.

2. The "ietf-udp-client" Module

The "ietf-udp-client" YANG module defines the "udp-client-grouping" grouping for configuring UDP clients with remote server information.

2.1. The "udp-client-grouping" Grouping

The following tree diagram [RFC8340] illustrates the tree structure of 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. YANG Module

This module imports types defined in [RFC6991].

```
<CODE BEGINS> file "ietf-udp-client@2024-02-26.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";
  }

  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.

    Copyright (c) 2024 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, is permitted pursuant to, and subject to the license
    terms contained in, the Revised BSD License set forth in Section
    4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
    (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 2024-02-26 {
```

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

    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
          "Specifies an IP address of the UDP client, which can be an
          IPv4 address or an IPv6 address.";
      }

      leaf remote-port {
        type inet:port-number;
        default "0";
        description
          "Specifies a port number of the UDP client. An invalid default
          value is used so that importing modules may 'refine' it with
          the appropriate default port number value.";
      }
    }
  }
  <CODE ENDS>
```

3. The "ietf-udp-server" Module

The "ietf-udp-server" YANG module defines the "udp-server-grouping" grouping for configuring UDP servers.

3.1. The "udp-server-grouping" Grouping

The following tree diagram [RFC8340] illustrates the structure of "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. YANG Module

The "ietf-udp-server" imports types defined in [RFC6991].

```
<CODE BEGINS> file "ietf-udp-server@2024-02-26.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";
  }

  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.

    Copyright (c) 2024 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, is permitted pursuant to, and subject to the license
    terms contained in, the Revised BSD License set forth in Section
    4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
    (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 2024-02-26 {
    description
      "Initial revision";
    reference
      "RFC-to-be: YANG Grouping for UDP Clients and UDP Servers";
  }
```

```
    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
          "Specifies an IP address of the UDP server, which can be an
          IPv4 address or an IPv6 address.";
      }

      leaf local-port {
        type inet:port-number;
        default "0";
        description
          "Specifies a port number of the UDP server. An invalid default
          value is used so that importing modules may 'refine' it with
          the appropriate default port number value.";
      }
    }
  }
  <CODE ENDS>
```

4. Security Considerations

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

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

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

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

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

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

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
maintained by IANA? N
reference: RFC-to-be

name: ietf-udp-server
namespace: urn:ietf:params:xml:ns:yang:ietf-udp-server
prefix: udps
maintained by IANA? N
reference: RFC-to-be

6. Acknowledgements

The authors would like to thank Mohamed Boucadair, Benoit Claise, Qiufang Ma and Qin Wu 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>.

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

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

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

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

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

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

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

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

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

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

[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>.

7.2. Informative 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-39, 22 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-39>.

[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>.

[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>.

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

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 20 April 2025

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

17 October 2024

YANG Groupings for UDP Clients and UDP Servers
draft-ietf-netconf-udp-client-server-05

Abstract

This document defines two YANG 1.1 modules to support the configuration of UDP clients and UDP servers.

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 20 April 2025.

Copyright Notice

Table of Contents

1. Introduction

This document defines two YANG 1.1 [RFC7950] modules to support the configuration of UDP clients and UDP servers [RFC768]. The data models defined by these modules may be used directly (e.g., to define a specific UDP client or UDP server) or in conjunction with the configuration defined for higher level protocols that depend on UDP.

2. The "ietf-udp-client" Module

This section defines a YANG 1.1 module called "ietf-udp-client". This YANG module defines the "udp-client" grouping for configuring UDP clients with remote server information.

Section 2.1 provides an overview of the YANG module for configuring UDP clients. An example of usage is illustrated in Section 2.2 and Section 2.3 defines the YANG module itself.

2.1. Data Model Overview

This section provides an overview of the features and the grouping defined in the "ietf-udp-client" YANG module.

2.1.1. Features

The "ietf-udp-client" module defines only one "feature" statements:

```
Features:
  +-- local-binding
```

This "local-binding" feature indicates that the client supports configuring local bindings (i.e., the local address and local port) for UDP clients.

The diagram above uses syntax that is similar to but not defined in [RFC8340].

2.1.2. The "udp-client" Grouping

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

```
module: ietf-udp-client

  grouping udp-client:
    +-- remote-address    inet:host
    +-- remote-port?      inet:port-number
    +-- local-address?    inet:ip-address {local-binding}?
    +-- local-port?       inet:port-number {local-binding}?
```

Comments:

* The "remote-address", which is mandatory, may be configured as an IPv4 address, an IPv6 address, or a hostname.

* The "remote-port" is defined with neither a "default" nor a "mandatory" statement. YANG modules using this grouping SHOULD refine the grouping with a "default" statement, when the port number is well-known (e.g., a port number allocated by IANA), or with a "mandatory" statement, if a port number needs to always be configured. This MAY be ignored when the port number is neither well-known nor mandatory to configure, such as might be the case when this grouping is used by another grouping.

* The "local-address", which is enabled by the "local-binding" feature, may be configured as an IPv4 address, an IPv6 address, or a wildcard value.

* The "local-port", which is enabled by the "local-binding" feature, is not mandatory. Its default value is "0", indicating that the operating system can pick an arbitrary port number.

2.2. Example Usage

This section presents an example of usage of the "udp-client" grouping.

```
<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<udp-client xmlns="urn:ietf:params:xml:ns:yang:ietf-udp-client">
  <remote-address>www.example.com</remote-address>
  <remote-port>10000</remote-port>
  <local-address>192.0.2.2</local-address>
  <local-port>12345</local-port>
</udp-client>
```

2.3. YANG Module

This module imports types defined in [RFC6991].

```
<CODE BEGINS> file "ietf-udp-client@2024-10-15"
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";
  }

  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.

      Copyright (c) 2024 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

      Redistribution and use in source and binary forms, with or without
      modification, is permitted pursuant to, and subject to the license
      terms contained in, the Revised BSD License set forth in Section
      4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
      (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 2024-10-15 {
      description
        "Initial revision";
      reference
        "RFC-to-be: YANG Groupings for UDP Clients and UDP Servers";
    }

    feature local-binding {
      description
        "Indicates that the UDP client supports configuring local
         bindings (i.e., the local address and local port) for
         UDP clients.";
    }

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

        Note that this grouping uses fairly typical descendant
        node names such that a stack of 'uses' statements will
        have name conflicts.  It is intended that the consuming
        data model will resolve the issue (e.g., by wrapping
        the 'uses' statement in a container called
        'udp-client-parameters').  This model purposely does
        not do this itself so as to provide maximum flexibility
        to consuming models.";

      leaf remote-address {
        type inet:host;
        mandatory true;
        description
          "The IP address or hostname of the remote UDP server.
          If a domain name is configured, then the DNS resolution should
```

```
            happen on each connection attempt. If the DNS resolution
            results in multiple IP addresses, the IP addresses
            are tried according to local preference order until
            a connection has been established or until all IP
            addresses have failed.";
        }

        leaf remote-port {
          type inet:port-number;
          description
            "The port number of the remote UDP server.";
        }

        leaf local-address {
          if-feature "local-binding";
          type inet:ip-address;
          description
            "The local IP address to bind to when sending UDP
            messages to the remote server. INADDR_ANY ('0.0.0.0') or
            INADDR6_ANY ('0:0:0:0:0:0:0:0' a.k.a. '::') may be used
            so that the server can bind to any IPv4 or IPv6 address.";
        }

        leaf local-port {
          if-feature "local-binding";
          type inet:port-number;
          default "0";
          description
            "The local port number to bind to when sending UDP
            messages to the remote server. The port number '0',
            which is the default value, indicates that any available
            local port number may be used.";
        }
      }
    }
    <CODE ENDS>
```

3. The "ietf-udp-server" Module

This section defines a YANG 1.1 module called "ietf-udp-server". This YANG module defines the "udp-server" grouping for configuring UDP servers.

Section 3.1 provides an overview of the YANG module for configuring UDP servers. An example of usage is illustrated in Section 3.2 while Section 3.3 defines the YANG module itself.

3.1. Data Model Overview

This section provides an overview of the grouping defined in the "ietf-udp-server" module.

3.1.1. The "udp-server" Grouping

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

```
module: ietf-udp-server

  grouping udp-server:
    +-- local-bind* [local-address]
       +-- local-address    inet:ip-address
       +-- local-port?      inet:port-number
```

Comments:

* The "local-address", which is mandatory, may be configured as an IPv4 address, an IPv6 address, or a wildcard value.

* The "local-port" is defined with neither a "default" nor a "mandatory" statement. YANG modules using this grouping SHOULD refine the grouping with a "default" statement, when the port number is well-known (e.g., a port number allocated by IANA), or with a "mandatory" statement, if a port number needs to always be configured. This MAY be ignored when the port number is neither well-known nor mandatory to configure, such as might be the case when this grouping is used by another grouping.

3.2. Example Usage

This section presents two examples of usage of the "udp-server" grouping.

This following shows an example of a server configured for listening to an IPv4 address:

```
<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<udp-server xmlns="urn:ietf:params:xml:ns:yang:ietf-udp-server">
  <local-bind>
    <local-address>192.0.2.2</local-address>
    <local-port>49152</local-port>
  </local-bind>
</udp-server>
```

This example shows an example of a server configured to listen to an IPv4 and IPv6 together:

```
<!-- The outermost element below doesn't exist in the data model. -->
<!--  It simulates if the "grouping" were a "container" instead.  -->

<udp-server xmlns="urn:ietf:params:xml:ns:yang:ietf-udp-server">
  <local-bind>
    <local-address>192.0.2.2</local-address>
    <local-port>49152</local-port>
  </local-bind>
  <local-bind>
    <local-address>2001:db8::0</local-address>
    <local-port>49153</local-port>
  </local-bind>
</udp-server>
```

3.3. YANG Module

The "ietf-udp-server" imports types defined in [RFC6991].

```
<CODE BEGINS> file "ietf-udp-server@2024-10-15.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";
  }

  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.

     Copyright (c) 2024 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.
```

```
        Redistribution and use in source and binary forms, with or without
        modification, is permitted pursuant to, and subject to the license
        terms contained in, the Revised BSD License set forth in Section
        4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
        (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 2024-10-15 {
      description
        "Initial revision";
      reference
        "RFC-to-be: YANG Groupings for UDP Clients and UDP Servers";
    }

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

         Note that this grouping uses fairly typical descendant
         node names such that a stack of 'uses' statements will
         have name conflicts.  It is intended that the consuming
         data model will resolve the issue (e.g., by wrapping
         the 'uses' statement in a container called
         'udp-server-parameters').  This model purposely does
         not do this itself so as to provide maximum flexibility
         to consuming models.";
      list local-bind {
        key "local-address";
        min-elements 1;
        description
          "A list of bind (listen) points for this server
           instance.  A server instance may have multiple
           bind points to support, e.g., the same port in
           different address families or different ports
           in the same address family.";
        leaf local-address {
          type inet:ip-address;
          mandatory true;
          description
            "The local IP address to listen on for incoming
             UDP messages.  To configure listening
             on all IPv4 addresses the value must be '0.0.0.0'
             (INADDR_ANY).  To configure listening on all IPv6
             addresses the value must be '::' (INADDR6_ANY).";
        }
        leaf local-port {
```

```
            type inet:port-number;
            description
              "The local port number to listen on for incoming UDP
               messages.";
          }
        }
      }
    }
    <CODE ENDS>
```

4. Security Considerations

This section uses the template described in Section 3.7 of [I-D.ietf-netmod-rfc8407bis].

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]. These network management protocols are required to use a secure transport layer and mutual authentication, e.g., SSH [RFC6242] without the "none" authentication option, Transport Layer Security (TLS) [RFC8446] with mutual X.509 authentication, and HTTPS with HTTP authentication (Section 11 of [RFC9110]).

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

The YANG module defines a set of identities, types, and groupings. These nodes are intended to be reused by other YANG modules. The module by itself does not expose any data nodes that are writable, data nodes that contain read-only state, or RPCs. As such, there are no additional security issues related to the YANG module that need to be considered.

Modules that use the groupings that are defined in this document should identify the corresponding security considerations. For example, reusing some of these groupings will expose privacy-related information (e.g., 'node-example').

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 URIs 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 IANA to register the following YANG modules in the YANG Module Names registry [RFC6020] within the "YANG Parameters" registry group:

name: ietf-udp-client
namespace: urn:ietf:params:xml:ns:yang:ietf-udp-client
prefix: udpc
maintained by IANA? N
reference: RFC-to-be

name: ietf-udp-server
namespace: urn:ietf:params:xml:ns:yang:ietf-udp-server
prefix: udps
maintained by IANA? N
reference: RFC-to-be

6. Acknowledgements

The authors would like to thank Mohamed Boucadair, Benoit Claise, Qiufang Ma and Qin Wu 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>.

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

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

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

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

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

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

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

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

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

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

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

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

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

7.2. Informative References

[I-D.ietf-netmod-rfc8407bis] Bierman, A., Boucadair, M., and Q. Wu, "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc8407bis-18, 11 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-rfc8407bis-18>.

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: 25 July 2024

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

22 January 2024

UDP-based Transport for Configured Subscriptions
draft-ietf-netconf-udp-notif-12

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 25 July 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.

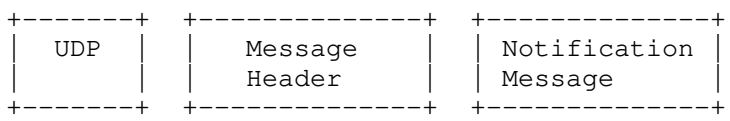


Figure 1: UDP-Notif Message Overview

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.

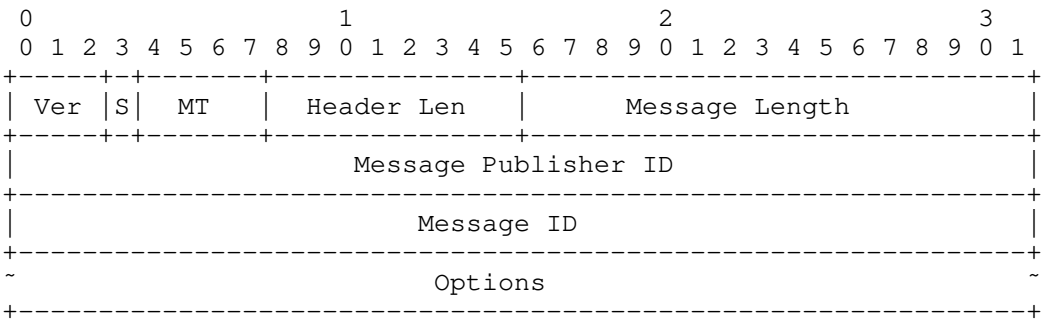


Figure 2: UDP-Notif Message Header Format

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. When S is set, the Private Encoding Option defined in Section 4.2 SHOULD be present in the UDP-notif message header.

* 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 are 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. When the last value (2^32-1) of Message ID has been generated, the Message ID wraps around and restarts at 0. 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.

All the binary fields MUST be encoded in network byte order (big endian).

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.

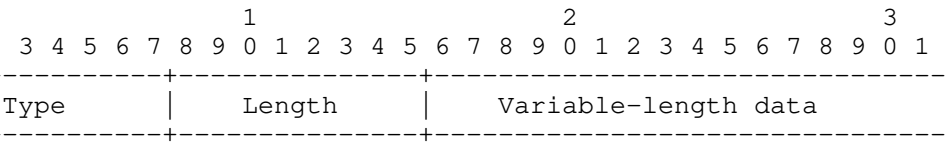


Figure 3: Generic Option Format

* 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 65527 bytes (65535 - 8 bytes). 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.

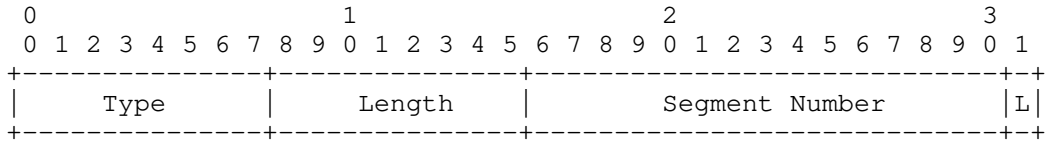


Figure 4: Segmentation Option Format

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. The Segment Number cannot wrap around.

* 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. The implementor or user SHOULD take into account the IP layer header size when setting the max-segment-size parameter to avoid fragmentation at the IP layer.

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.

The receiver SHOULD support the reception of unordered segments. The implementation of the receiver SHOULD provide an option to discard the received segments if, after some time, one of the segments is still missing and the reassembly of the message is not possible.

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.

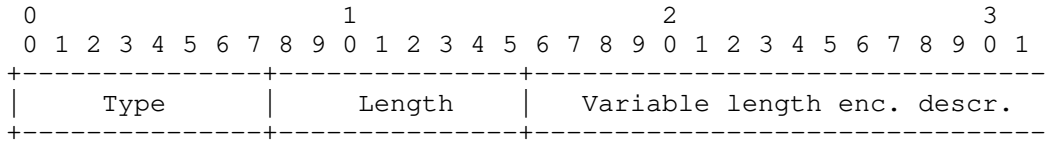


Figure 5: Private Encoding Option Format

* 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.

It is RECOMMENDED that the size of a Notification Message is small and segmentation does not result in segmenting the message into too much segments to avoid dropping the entire message when there is a lost segment. When a Notification Message is large, it is RECOMMENDED to use a reliable transport such as HTTPS-notif [I-D.ietf-netconf-https-notif].

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 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. YANG to configure UDP-notif

The YANG model described in Section 7.2 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}?
```

7.2. 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 from the "ietf-udp-client" module [I-D.ahuang-netconf-udp-client-server] and the "tls-client-grouping" defined in the "ietf-tls-client" module [I-D.ietf-netconf-tls-client-server].

<CODE BEGINS> file "ietf-udp-notif-transport@2024-01-22.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 ZZZZ: YANG Grouping for UDP Clients and UDP Servers";
  }
  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:  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 2024-01-22 {
      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 dtls13 {
      description
        "This feature indicates that DTLS 1.3 encryption of UDP
         packets 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;

      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.";
      }
```

```
      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-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-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 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, Michael Tuxen, 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.ahuang-netconf-udp-client-server]
Feng, A. H., Francois, P., and K. Watsen, "YANG Grouping for UDP Clients and UDP Servers", Work in Progress, Internet-Draft, draft-ahuang-netconf-udp-client-server-01, 22 January 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ahuang-netconf-udp-client-server/>.

[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-14, 18 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-14>.

[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-34, 28 December 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-34>.

[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-04, 22 January 2024, <https://datatracker.ietf.org/api/v1/doc/document/draft-ahuang-netconf-notif-yang/>.

[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

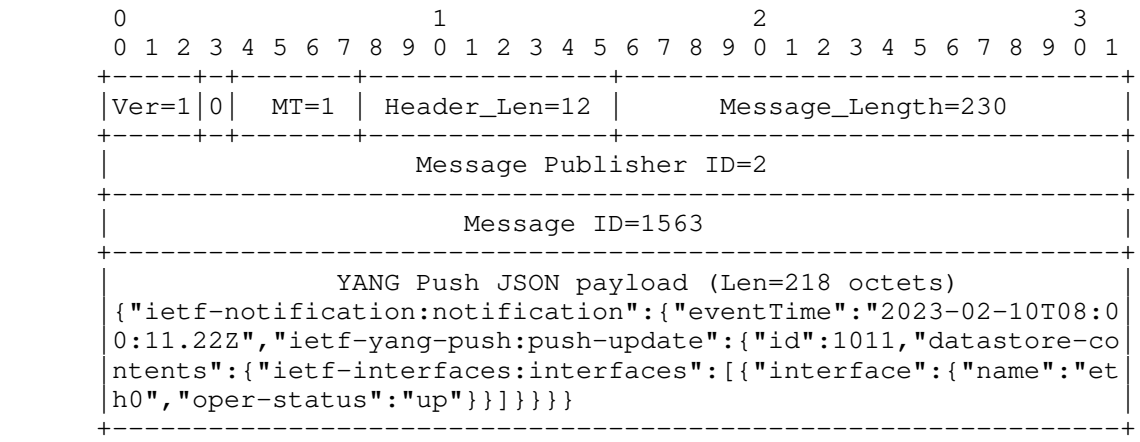


Figure 7: UDP-notif transport message

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

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 4 September 2024

Z. Lin
B. Claise
Huawei
I. D. Martinez-Casanueva
Telefonica Innovacion Digital

3 March 2024

Augmented-by Addition into the IETF-YANG-Library
draft-lincla-netconf-yang-library-augmentation-01

Abstract

This document augments the ietf-yang-library in [RFC8525] to provide the augmented-by list. It facilitates the process of obtaining the entire dependencies of YANG model, by directly querying the server's YANG module.

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/Zephyre777/draft-lincla-netconf-yang-library-augmentation.

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 4 September 2024.

Table of Contents

1. Introduction

The YANG library [RFC8525] specifies a YANG module that provides the information about the YANG models and datastores to facilitate a client application to fully utilize and understand the YANG data modelling language. To know the YANG dependencies, [RFC8525] has defined and provided the submodule list and the YANG modules deviation list. However, the YANG modules augmentation is not provided.

According to [RFC7950], both augmentations and deviations are defining contents external to the model, but applying internally for the model. It is important to know the augmentation and deviation as they are dependencies of the model, but it is also difficult because they are defined externally. When we try to use the ietf-yang-library in [RFC8525] to obtain the reverse dependencies (Augmentations and deviations), the augmentation is not defined in it.

However, the augmentation and the deviation work similarly as YANG modules dependency. Therefore, it is reasonable to document them the same way in the IETF YANG library. Besides, it will be easier to determine the reverse dependency if the augmentation is directly available, through a GET request into this new YANG model.

This draft augment the ietf-yang-library to include the YANG modules augmentation information.

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 terminology from [RFC8525] is used in this document

Tree diagrams in this document use the notation defined in [RFC8340] .

2. Motivation

When using one YANG model, it is important to make sure that all its dependencies are presented. In [RFC7950] there are four dependencies for one YANG mode:

* Import: the "import" statement allows a module or submodule to reference definitions defined in other modules.

* Include: the "include" statement is used in a module to identify each submodule that belongs to it.

* Augmentation: the "augment" statement defines the location in the data model hierarchy where additional nodes are inserted

* Deviation: the "deviation" statement defines a hierarchy of a module that the server does not implement faithfully.

The import and include are direct dependencies while the augmentation and deviation are reverse dependencies. To know a specific YANG model's direct dependencies, we can parse this YANG model as the dependencies are directly specified (import and include statements").

As for the reverse dependencies, since they are defined externally, we cannot parse the YANG model itself to get them. Among all the methods for getting reverse dependency, getting the ietf-yang-library content is one of the most convenient ways.

However, the ietf-yang-library only provides the deviation list, but not the augmentation. It is reasonable to update it to provide the augmentation information since both augmentation and deviation have similar way of working (both are applied to the original model but invisible to them). A noticeable difference between deviations and augmentations is that the deviations are required to understand the API contract between the client and the server. But with some use cases arise as the requirement of automate network management, the augmentation becomes essential information for understanding the network, too.

3. Use Cases

As the demand arises for YANG-based telemetry [RFC8641], there is a need for real-time knowledge of a specific YANG model's dependency list when a specific YANG-Push message is received.

The alternative, for a YANG-push receiver, to collect and store the entire module set for every single server who could be streaming data, is not always practical. See the following reasons:

* For a YANG-push collector => we never know in advance which telemetry content will be received and from whom.

* Querying all the YANG modules is time consuming => we lose the real-time.

With similar central idea, two use cases are introduced in this section. One target solving the dependencies problems in a Data Mesh Telemetry System while the other aims at building a data catalog which makes YANG model inforamtion easily accessible.

3.1. Data Mesh Telemetry Architecture

A network analytics architecture that integrates YANG-push and Kafka is proposed in 2022 and is continuously growing and gaining influence, refer to the draft: An Architecture for YANG-Push to Apache Kafka Integration [I-D.netana-nmop-yang-kafka-integration].

In this open-sourced project covering Support of Versioning in YANG Notifications Subscription [I-D.ietf-netconf-yang-notifications-versioning], Support of Network Observation Timestamping in YANG Notifications [I-D.netconf-tgraf-yang-push-observation-time], among others, the purpose is to provide adequate information in the YANG-Push notification so that when it is received, the model and its dependency can be parsed and found automatically from the vantage point. The architecture relies on the information of YANG model and their dependency to realize, as one of its main goals is to solve the problem of missing YANG semantics when data is received in Time Series Database in the end. To solve the problem, a schema registry is introduced to store YANG models and all their relationship (direct dependency and reverse dependency).

Currently, the method used for finding model's reverse dependency is get-all-schemas, that is to retrieve all YANG modules from the device to the client's disk, enabling the client to fully understand the YANG model relationship. This process is heavy because in real situation, each device may have few thousands or even more YANG module implemented. Besides, pressure is also introduced due to the need of parsing modules and find all the dependencies.

Considering the telemetry real-time aspects, this extra delay in processing the dependencies through get-all-schemas is not ideal.

The YANG model proposed in the draft can be used in this architecture to release the stress of get-all-schemas and bring some extra benefits.

By providing the augmentation information, it enables the collector to get the YANG reverse dependencies by simply sending one <get> query to ietf-yang-library. In this regard, the process of acquiring full dependency becomes real-time action.

Compared with get-all-schemas, it enables collector to fetch up-to-date data since the queries are sent at run time, while the old approach forces collector to always work with never updated data. On another hand, user do not bother waiting for ten minutes every time when starting collector to either get data updated or to obtain YANG relationship.

3.2. Data Catalog

Finding the YANG models implemented by a network device is paramount for configuring and monitoring the status of a network. However, since the inception of YANG the network industry has experienced a tsunami of YANG models developed by SDOs, open-source communities, and network vendors. This heterogeneity of YANG models, that vary from one network device model to another, makes the management of a multi-vendor network a big challenge for operators. [Martinez-Casanueva2023]

In this regard, a data catalog provides a registry of the datasets exposed by remote data sources for consumers to discover data of interest. Besides the location of the dataset (i.e., the data source), the data catalog registers additional metadata such as the data model (or schema) followed in the dataset or even related terms defined in a business glossary.

Data catalog solutions typically implement collectors that ingest metadata from the data sources themselves and also external metadata sources. For example, a Kafka Schema Registry is a metadata source that provides metadata about the data models followed by some data stored in a Kafka topic.

In this sense, a YANG-enabled network device can be considered as another kind of data source, which the Data Catalog can pull metadata from. For instance, the data catalog can include a connector that fetches metadata about the YANG models implemented by the network device. Combining these metadata with other such as the business concept "interface", would enable data consumers to discover which datasets related to the concept "interface" are exposed by the network device.

Network devices that implement YANG Library expose metadata about which YANG modules are implemented, and which are only imported. However, what a data consumer needs at the end are the YANG models implemented by the device, hence, the combination of implemented YANG modules with other YANG modules that might deviate or augment the formers.

Coming back to the example of datasets related to the "interface" concept, say we have a network device that implements the ietf-interfaces module [RFC8343] and the ietf-ip module [RFC8344], where the latter augments the former. For a data catalog to collect these metadata, a connector would retrieve YANG Library data from the target device. However, the current version of YANG Library would not satisfy the use case as it would tell that the device implements both ietf-interfaces and ietf-ip modules, but will miss the augment dependency between them.

The current workaround to this limitation is to, in combination with the YANG library data, additionally fetch both YANG modules and process them to discover that there is an augment dependency. This adds extra burden on the connector, which is forced to combine multiple metadata collection mechanisms. This process could be softened by extending YANG Library to also capture augment dependencies, in a similar fashion to deviation dependencies.

4. The "ietf-yang-library-augmentedby" YANG module

This YANG module augments the ietf-yang-library module by adding the augmented-by list in the "yang-library/module-set". The name Augmented-by indicated the modules by which the current module is being augmented. Note that this module only augments the ietf-yang-library defined in [RFC8525]. At the time of writing this document, most router vendors support [RFC7895], a previous revision of the ietf-yang-library YANG module; The module that augments [RFC7895] is provided in the appendix B.

4.1. Data Model Overview

4.1.1. Tree View

The following is the YANG tree diagram for model ietf-yang-library-augmentedby.

```
module: ietf-yang-library-augmentedby

  augment /yanglib:yang-library/yanglib:module-set/yanglib:module:
    +--ro augmented-by*   -> ../../yanglib:module/name
```

4.1.2. Full Tree View

The following is the YANG tree diagram[RFC8340] for the ietf-yang-library with the augmentation defined in module ietf-yang-library-augmentedby, including the RPCs and notifications.

```
   module: ietf-yang-library
     +--ro yang-library
     |  +--ro module-set* [name]
     |  |  +--ro name                  string
     |  |  +--ro module* [name]
     |  |  |  +--ro name                          yang:yang-identifier
     |  |  |  +--ro revision?                     revision-identifier
     |  |  |  +--ro namespace                     inet:uri
     |  |  |  +--ro location*                     inet:uri
     |  |  |  +--ro submodule* [name]
     |  |  |  |  +--ro name        yang:yang-identifier
     |  |  |  |  +--ro revision?   revision-identifier
     |  |  |  |  +--ro location*   inet:uri
     |  |  |  +--ro feature*                      yang:yang-identifier
     |  |  |  +--ro deviation*                    -> ../../module/name
     |  |  |  +--ro yanglib-aug:augmented-by*
     |  |  |                                -> ../../yanglib:module/name
     |  |  +--ro import-only-module* [name revision]
     |  |     +--ro name         yang:yang-identifier
     |  |     +--ro revision     union
     |  |     +--ro namespace    inet:uri
     |  |     +--ro location*    inet:uri
     |  |     +--ro submodule* [name]
     |  |        +--ro name        yang:yang-identifier
     |  |        +--ro revision?   revision-identifier
     |  |        +--ro location*   inet:uri
     |  +--ro schema* [name]
     |  |  +--ro name          string
     |  |  +--ro module-set*   -> ../../module-set/name
     |  +--ro datastore* [name]
     |  |  +--ro name      ds:datastore-ref
     |  |  +--ro schema    -> ../../schema/name
     |  +--ro content-id    string
     x--ro modules-state
        x--ro module-set-id    string
        x--ro module* [name revision]
           x--ro name                yang:yang-identifier
           x--ro revision            union
           +--ro schema?             inet:uri
           x--ro namespace           inet:uri
           x--ro feature*            yang:yang-identifier
           x--ro deviation* [name revision]
           |  x--ro name        yang:yang-identifier
           |  x--ro revision    union
           x--ro conformance-type    enumeration
           x--ro submodule* [name revision]
              x--ro name        yang:yang-identifier
              x--ro revision    union
```

```
             +--ro schema?     inet:uri

     notifications:
       +---n yang-library-update
       |  +--ro content-id    -> /yang-library/content-id
       x---n yang-library-change
          x--ro module-set-id    -> /modules-state/module-set-id
```

4.1.3. YANG Module

The YANG module source code of ietf-yang-library-augmentedby in which augmentation to the ietf-yang-library of [RFC8525] is defined.

```
<CODE BEGINS> file "ietf-yang-library-augmentedby@2023-10-27.yang"
module ietf-yang-library-augmentedby {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-library-augmentedby";
  prefix yanglib-aug;

  import ietf-yang-library {
    prefix yanglib;
    reference
      "RFC 8525: YANG library";
  }

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

     Author:   Zhuoyao Lin
               <mailto:zephyre888@gmail.com>
               Benoit Claise
               <mailto:benoit.claise@huawei.com>
               IGNACIO DOMINGUEZ MARTINEZ-CASANUEVA
               <matilto:ignacio.dominguezmartinez@telefonica.com>";

  description
    "This module augments the ietf-yang-library defined in
     [RFC8525] to provide not only the deviation list, but also
     the augmented-by list, in order to give sufficient
     information about the YANG models reverse dependency. It
     facilitates the process of obtaining the entire
     dependencies of YANG model.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
     'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
```

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

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

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

    revision 2023-10-27 {
      description
        "Added list augmented-by in yang-library/module-set/module to
        make the module store the entire reverse dependency information
        (augmented-by and deviation).";
      reference
        "RFC XXXX: Support of augmentedby in ietf-yang-library";
    }

    augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
      description
        "Augment the augmented-by list from module info with the
        module-augmented-by grouping" ;

      leaf-list augmented-by {
        type leafref {
          path "../../yanglib:module/yanglib:name";
        }

        description
          "Leaf-list of the augmentation used by this server to
           modify the conformance of the module associated with
           this entry.  Note that the same module can be used for
           augmented-by for multiple modules, so the same
           entry MAY appear within multiple 'module' entries.

           This reference MUST NOT (directly or indirectly)
           refer to the module being augmented.

           Robust clients may want to make sure that they handle a
           situation where a module augments itself (directly or
```

```
            indirectly) gracefully.";
        }
      }
    }
    <CODE ENDS>
```

5. Implementation Status

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

5.1. draft repository

Here is the github repository for the YANG source code of this draft: https://github.com/Zephyre777/draft-lincla-netconf-yang-library-augmentation.git.

For the demo, please refer to the demo folder in this repository. There is a netopeer2 instance running with updated version of libyang and sysrepo(links are listed in the demo instruction).

6. Changes

6.1. Changes from 00 to 01

The list name has been updated from "augmentation" to "augmented-by", in order to represent the usage clearly.

The leafref has been changed from absolute path "/yanglib:yang-libraray/yanglib:module-set/yanglib:module/yanglib:name" to relative path "../../yanglib:module/yanglib:name". The YANG validation in the appendix A shows that this path can work as expected.

Section 5 Implementation and section 6 Changes has been added.

7. Security Considerations

TBC

8. IANA Considerations

This document has no actions for IANA.

9. References

9.1. Normative References

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

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <https://www.rfc-editor.org/info/rfc7895>.

[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>.

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

[RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <https://www.rfc-editor.org/info/rfc8344>.

[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>.

9.2. Informative References

[I-D.ietf-netconf-yang-notifications-versioning] Graf, T., Claise, B., and A. H. Feng, "Support of Versioning in YANG Notifications Subscription", Work in Progress, Internet-Draft, draft-ietf-netconf-yang-notifications-versioning-03, 20 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-yang-notifications-versioning-03>.

[I-D.netana-nmop-yang-kafka-integration]
Graf, T., "An Architecture for YANG-Push to Apache Kafka Integration", Work in Progress, Internet-Draft, draft-netana-nmop-yang-kafka-integration-00, 24 February 2024, <https://datatracker.ietf.org/doc/html/draft-netana-nmop-yang-kafka-integration-00>.

[I-D.netconf-tgraf-yang-push-observation-time]
Graf, T., Claise, B., and A. H. Feng, "Support of Network Observation Timestamping in YANG Notifications", Work in Progress, Internet-Draft, draft-netconf-tgraf-yang-push-observation-time-00, 6 July 2023, <https://datatracker.ietf.org/doc/html/draft-netconf-tgraf-yang-push-observation-time-00>.

[Martinez-Casanueva2023]
Martinez-Casanueva, I. D., Gonzalez-Sanchez, D., Bellido, L., Fernandez, D., and D. R. Lopez, "Toward Building a Semantic Network Inventory for Model-Driven Telemetry", DOI 10.1109/MCOM.001.2200222, March 2023, <https://doi.org/10.1109/MCOM.001.2200222>. IEEE Communications Magazine

[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>.

Appendix A. YANG module validation with yanglint

This section gives a few examples that the user can try themselves with yanglint. This is created to prove the syntax correctness.

The valid example should pass the validation while the invalid one will not, because the module has augmented a module in another module-set, which is illegal according to the YANG source code.

A.1. A valid ietf-yang-library data example

```
<CODE BEGINS> file "example_valid.xml"
<yang-library xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
  <content-id>1</content-id>
  <module-set>
    <name>ms1</name>
    <module>
      <name>module1</name>
      <revision>2024-02-29</revision>
      <namespace>urn:ietf:params:xml:ns:yang:module1</namespace>
      <augmented-by
      xmlns="urn:ietf:params:xml:ns:yang:
      ietf-yang-library-augmentedby">module2</augmented-by>
      <augmented-by
      xmlns="urn:ietf:params:xml:ns:yang:
      ietf-yang-library-augmentedby">module3</augmented-by>
    </module>
    <module>
      <name>module2</name>
      <revision>2024-02-29</revision>
      <namespace>urn:ietf:params:xml:ns:yang:module2</namespace>
    </module>
    <module>
      <name>module3</name>
      <revision>2024-02-29</revision>
      <namespace>urn:ietf:params:xml:ns:yang:module3</namespace>
    </module>
  </module-set>
</yang-library>
<modules-state xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
    <module-set-id>0</module-set-id>
</modules-state>
<CODE ENDS>
```

A.2. An invalid ietf-yang-library data example

```
<CODE BEGINS> file "example_invalid.xml"
<yang-library xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
  <content-id>1</content-id>
  <module-set>
    <name>ms1</name>
    <module>
      <name>module1</name>
      <revision>2024-02-29</revision>
      <namespace>urn:ietf:params:xml:ns:yang:module1</namespace>
      <augmented-by
      xmlns="urn:ietf:params:xml:ns:yang:
      ietf-yang-library-augmentedby">module2</augmented-by>
      <augmented-by
      xmlns="urn:ietf:params:xml:ns:yang:
      ietf-yang-library-augmentedby">module3</augmented-by>
    </module>
    <module>
      <name>module2</name>
      <revision>2024-02-29</revision>
      <namespace>urn:ietf:params:xml:ns:yang:module2</namespace>
    </module>
    <module>
      <name>module3</name>
      <revision>2024-02-29</revision>
      <namespace>urn:ietf:params:xml:ns:yang:module3</namespace>
    </module>
  </module-set>
</yang-library>
<modules-state xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
    <module-set-id>0</module-set-id>
</modules-state>
<CODE ENDS>
```

Appendix B. YANG Module augmenting RFC7895

This section defines the ietf-yang-library-rfc7895-augmentedby that augments the ietf-yang-library defined in [RFC7895]. The module-state/module list of this YANG module version is also defined in the [RFC8525] version though deprecated.

B.1. Tree View for YANG module augmenting RFC7895

The following is the YANG tree diagram for ietf-yang-library-rfc7895-augmentedby augmenting RFC7895.

```
   module: ietf-yang-library-rfc7895-augmentedby

     augment /yanglib:modules-state/yanglib:module:
       x--ro augmentedby* [name revision]
          +--ro name        -> /yanglib:modules-state/module/name
          +--ro revision    -> /yanglib:modules-state/module/revision
```

B.2. Full Tree View for ietf-yang-library with augmentation to RFC7895

The following is the full YANG tree diagram of ietf-yang-library-rfc7895-augmentedby augmenting ietf-yang-library defined in RFC7895.

```
   module: ietf-yang-library
     +--ro modules-state
        +--ro module-set-id    string
        +--ro module* [name revision]
           +--ro name                          yang:yang-identifier
           +--ro revision                      union
           +--ro schema?                       inet:uri
           +--ro namespace                     inet:uri
           +--ro feature*                      yang:yang-identifier
           +--ro deviation* [name revision]
           |  +--ro name        yang:yang-identifier
           |  +--ro revision    union
           +--ro conformance-type              enumeration
           +--ro submodule* [name revision]
           |  +--ro name        yang:yang-identifier
           |  +--ro revision    union
           |  +--ro schema?     inet:uri
           x--ro yanglib-aug:augmented-by* [name revision]
              +--ro yanglib-aug:name
                           -> /yanglib:modules-state/module/name
              +--ro yanglib-aug:revision
                           -> /yanglib:modules-state/module/revision

     notifications:
       +---n yang-library-change
          +--ro module-set-id    -> /modules-state/module-set-id
```

B.3. YANG module augmenting RFC7895

The YANG module that augments the ietf-yang-library RFC7895.

```
   <CODE BEGINS>
     file "ietf-yang-library-rfc7895-augmentedby@2023-10-27.yang"
   module ietf-yang-library-rfc7895-augmentedby {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-yang-library-rfc7895-augmentedby
";
     prefix yanglib-aug;

     import ietf-yang-library {
       prefix yanglib;
       revision-date 2016-06-21;
       reference
         "RFC 7895: YANG Module Library.";
     }

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

        Author:   Zhuoyao Lin
                  <mailto:zephyre888@gmail.com>
        Author:   Benoit Claise
                  <mailto:benoit.claise@huawei.com>
        Author:   IGNACIO DOMINGUEZ MARTINEZ-CASANUEVA
                  <matilto:ignacio.dominguezmartinez@telefonica.com>";

     description
       "This module augments the ietf-yang-library defined in [RFC7895]
        to provide not only the deviation list, but also the
        augmentedby list, in order to give sufficient information
        about the YANG models reverse dependency. It facilitates
        the process of obtaining the entire dependencies of YANG model.

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

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

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Revised BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
```

```
        Relating to IETF Documents
        (https://trustee.ietf.org/license-info).
        This version of this YANG module is part of RFC XXXX; see the
        RFC itself for full legal notices.  ";


      revision 2023-10-27 {
        description
          "Added list augmentedby in yang-library/modules-state/module to
          make the module store the entire reverse dependency information
          (augmentedby and deviation).";
        reference
          "RFC XXXX: Support of augmentedby in ietf-yang-library
              defined in RFC7895";
      }

      augment "/yanglib:modules-state/yanglib:module" {
        description
          "Augment the augmentedby from module info with the
          module-augmented-by grouping" ;
        uses yanglib-aug:module-state-augmented-by;
      }

      /*
       * Groupings
       */

      grouping module-state-augmented-by {
        description
          "This grouping defines a list with keys being the module
          name and revison. The list contains the augmented-by list.";

        list augmented-by {
          key "name revision";
          status deprecated;

          description
            "List of YANG augmented-by module names and revisions
             used by this server to modify the conformance of
             the module associated with this entry.  Note that
             the same module can be used for augmented-by for
             multiple modules, so the same entry MAY appear
             within multiple 'module' entries.

             The augment module MUST be present in the 'module'
             list, with the same name and revision values.
             The 'conformance-type' value will be 'implement' for
             the augment module.";
```

```
        leaf name {
          type leafref {
            path "/yanglib:modules-state/yanglib:module/yanglib:name";
          }
          description
            "Identifies a given module in the yang library by
            its name.";
        }

        leaf revision {
          type leafref {
            path "/yanglib:modules-state/yanglib:module/yanglib:revision";
          }
          description
            "Revision of the module";
        }
      }
    }
  }
  <CODE ENDS>
```

Contributors

The following people all contributed to creating this document:

Acknowledgements

The author would like to thanks Jan Lindblad and Jean Quilbeuf for his help during the design of the YANG module.

Authors' Addresses

Zhuoyao
Huawei
Townsend Street, 4th Floor George's Court
Dublin
Ireland
Email: zephyre888@gmail.com

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

Ignacio Dominguez Martinez-Casanueva
Telefonica Innovacion Digital
Ronda de la Comunicacion, S/N
Madrid 28050
Spain
Email: ignacio.dominguezmartinez@telefonica.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 14 July 2024

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

11 January 2024

RESTCONF Extension to support Trace Context Headers
draft-netconf-restconf-trace-ctx-headers-00

Abstract

This document extends the RESTCONF protocol in order to support trace context propagation as defined by the W3C.

About This Document

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

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

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

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

Status of This Memo

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

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

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 14 July 2024.

Table of Contents

1. Introduction

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

The W3C has defined two HTTP headers (traceparent and tracestate) for context propagation that are useful for distributed systems like the ones defined in [RFC8309]. The goal of this document is to adopt this W3C specification for the RESTCONF protocol.

This document does not define new HTTP extensions but makes those defined in [W3C-Trace-Context] optional headers for the RESTCONF protocol.

In [I-D.draft-rogaglia-netconf-trace-ctx-extension-03], the NETCONF protocol extension is defined and we will re-use several of the YANG and XML objects defined in that document for RESTCONF. Please refer to that document for additional context and example applications.

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.

2. RESTCONF Extensions

A RESTCONF server SHOULD support trace context traceparent header as defined in [W3C-Trace-Context].

A RESTCONF server SHOULD support trace context tracestate header as defined in [W3C-Trace-Context].

2.1. Errors handling

The RESTCONF server SHOULD follow the "Processing Model for Working with Trace Context" as specified in [W3C-Trace-Context].

If the server rejects the RPC because of the trace context headers values, the server MUST return an rpc-error with the following values:

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

Additionally, the error-info tag SHOULD contain a relevant details about the error.

Finally, the sx:structure defined in [I-D.draft-rogaglia-netconf-trace-ctx-extension-03] SHOULD be present in any error message from the server.

Example of a badly formated trace context extension using [RFC8040] example B.2.1:

```
POST /restconf/data/example-jukebox:jukebox/library HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json
traceparent: SomeBadFormatHere
tracestate: OrSomeBadFormatHere

{
  "example-jukebox:artist" : [
    {
      "name" : "Foo Fighters"
    }
  ]
}
```

And the expected error message:

```
HTTP/1.1 400 Bad Request
Date: Tue, 20 Jun 2023 20:56:30 GMT
Server: example-server
Content-Type: application/yang-data+json

{ "ietf-restconf:errors" : {
    "error" : [
      {
        "error-type" : "protocol",
        "error-tag" : "operation-failed",
        "error-severity" : "error",
        "error-message" :
        "OTLP traceparent attribute incorrectly formatted",
        "error-info": {
          "ietf-netconf-otlp-context:meta-name" : "traceparent",
          "ietf-netconf-otlp-context:meta-value" :
          "SomeBadFormatHere",
          "ietf-netconf-otlp-context:error-type" :
          "ietf-netconf-otlp-context:bad-format"
        }
      }
    ]
  }
}
```

2.2. Trace Context header versionning

This extension refers to the [W3C-Trace-Context] trace context capability. The W3C traceparent and trace-state headers include the notion of versions. It would be desirable for a RESTCONF client to be able to discover the one or multiple versions of these headers supported by a server. We would like to achieve this goal avoiding the deffinition of new RESTCONF capabilities for each headers' version.

[I-D.draft-rogaglia-netconf-trace-ctx-extension-03] defines a pair YANG modules that SHOULD be included in the YANG library per [RFC8525] of the RESTCONF server supporting the RESTCONF Trace Context extension that will refer to the headers' supported versions. Future updates of this document could include additional YANG modules for new headers' versions.

3. Security Considerations

TODO Security

4. IANA Considerations

This document has no IANA actions.

5. Acknowledgments

We would like to acknowledge

6. References

6.1. Normative References

[I-D.draft-rogaglia-netconf-trace-ctx-extension-03]
Gagliano, R., Larsson, K., and J. Lindblad, "NETCONF Extension to support Trace Context propagation", Work in Progress, Internet-Draft, draft-rogaglia-netconf-trace-ctx-extension-03, 6 July 2023, <https://datatracker.ietf.org/doc/html/draft-rogaglia-netconf-trace-ctx-extension-03>.

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

[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>.

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

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

6.2. Informative References

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

Appendix A. Example RESTCONF calls

TBD

Appendix B. Changes (to be deleted by RFC Editor)

B.1. From version 00 to draft-netconf-restconf-trace-ctx-headers-00

* Adopted by NETCONF WG
* Moved repository to NETCONF WG
* Changed build system to use martinthomson’s excellent framework
* Ran make fix-lint to remove white space at EOL etc.
* Added this change note. No other content changes.

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

* Security Considerations
* Example RESTCONF Calls

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

Authors' Addresses

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

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

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 14 July 2024

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

11 January 2024

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

Abstract

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

About This Document

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

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

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

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

Table of Contents

1. Introduction

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

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

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

A trace context also differs from [I-D.ietf-netconf-transaction-id] in several ways as the trace operation may involve any operation (including for example validate, lock, unlock, etc.) Additionally, a trace context scope may include the full application stack (orchestrator, controller, devices, etc) rather than a single NETCONF server, which is the scope for the transaction-id. The trace context is also complemetary to [I-D.ietf-netconf-transaction-id] as a given trace-id can be associated with the different transaction-ids as part of the information exported to the collector.

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

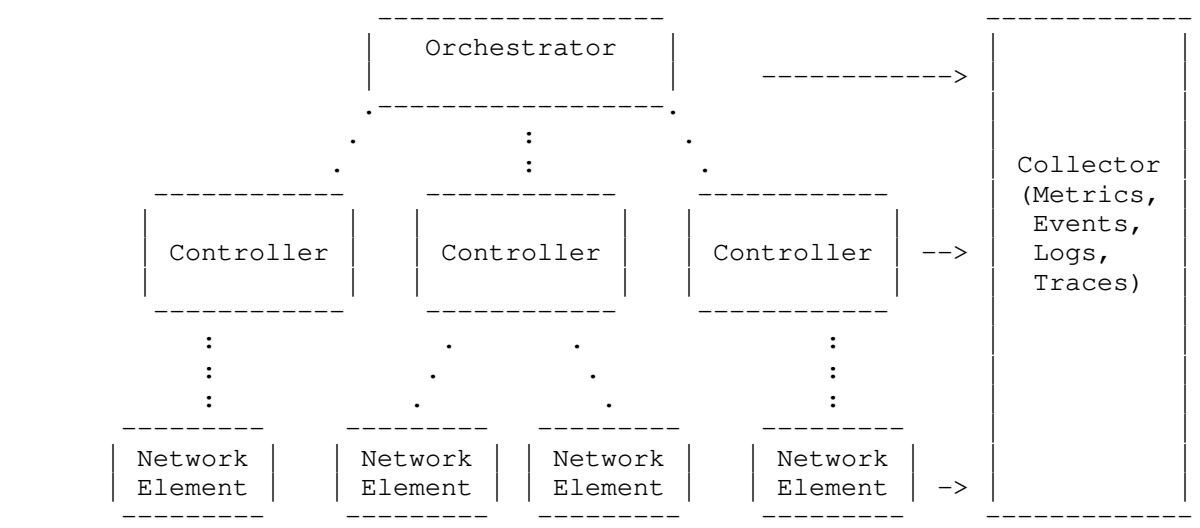


Figure 1: A Sample SDN Architecture from RFC8309 augmented to include the export of metrics, events, logs and traces from the different components to a common collector.

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

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

1.1. Implementation example 1: OpenTelemetry

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

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

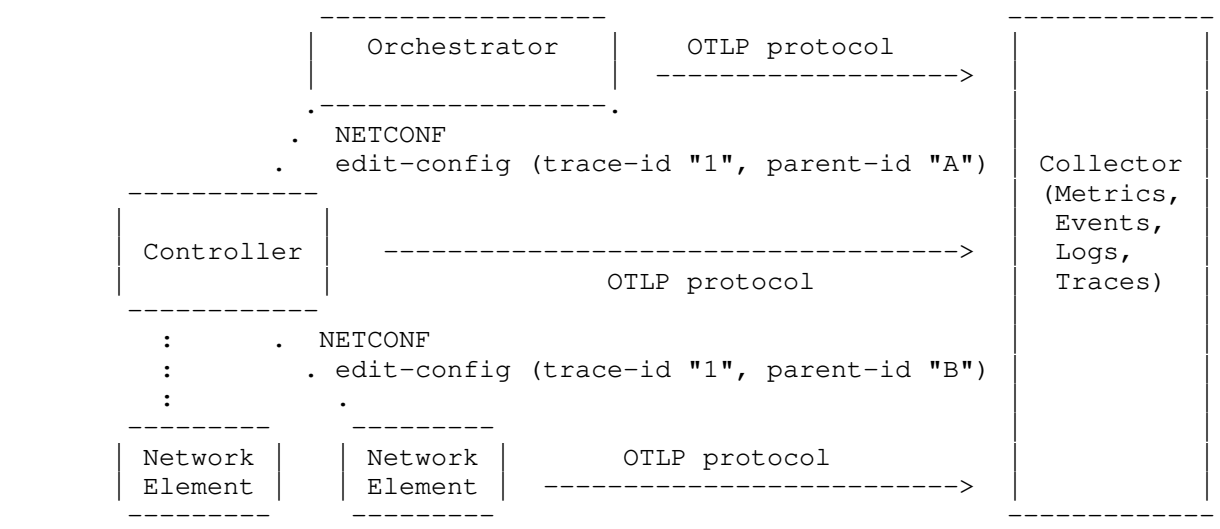


Figure 2: An implementation example where the NETCONF protocol is used between the Orchestrator and the Controller and also between the Controller and the Network Elements. Every component exports M.E.L.T information to the collector using the OTLP protocol.

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

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

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

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

1.2. Implementation example 2: YANG DataStore

OpenTelemetry implements the "push" model for data streaming where information is sent to the back-end as soon as produced and is not required to be stored in the system. In certain cases, a "pull" model may be envisioned, for example for performing forensic analysis while not all OTLP traces are available in the back-end systems.

An implementation of a "pull" mechanism for M.E.L.T. information in general and for traces in particular, could consist of storing traces in a yang datastore (particularly the operational datastore.) Implementations should consider the use of circular buffers to avoid resources exhaustion. External systems could access traces (and particularly past traces) via NETCONF, RESTCONF, gNMI or other polling mechanisms. Finally, storing traces in a YANG datastore enables the use of YANG-Push [RFC8641] or gNMI Telemetry as an additional "push" mechanisms.

This document does not specify the YANG module in which traces could be stored inside the different components. That said, storing the context information described in this document as part of the recorded traces would allow back-end systems to correlate the information from different components as in the OpenTelemetry implementation.

Note to be removed in the future: Some initial ideas are under discussion in the IETF for defining a standard YANG data model for traces. For example see: I-D.quilbeuf-opsawg-configuration-tracing which focusses only on configuration change root cause analysis use case (see the use case desciption below). These ideas are complementary to this draft.

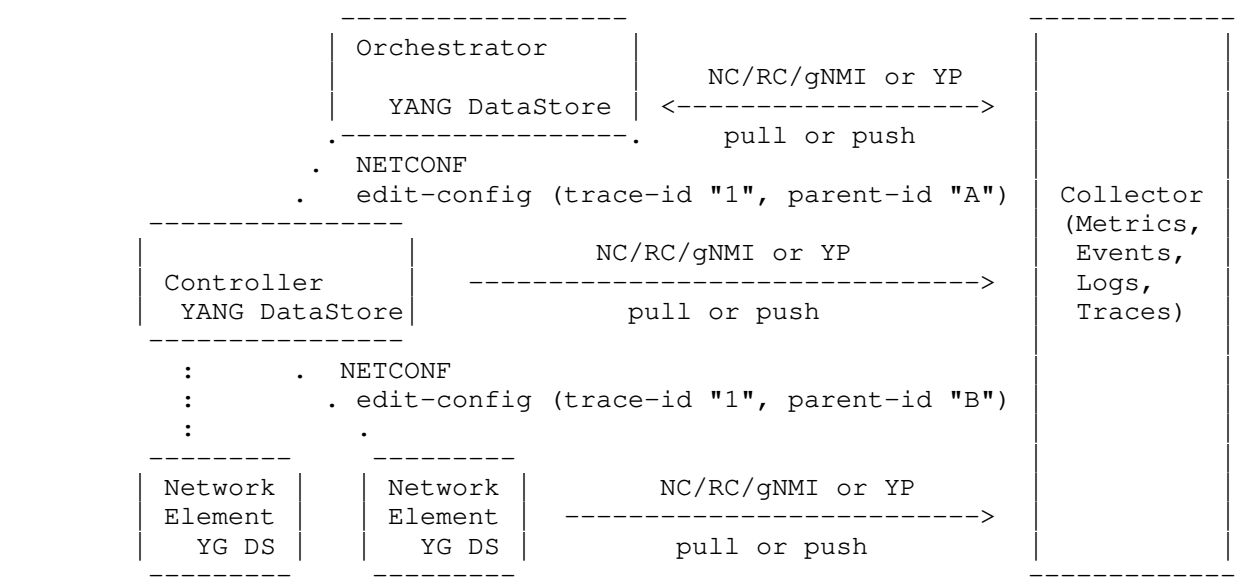


Figure 3: An implementation example where the NETCONF protocol is used between the Orchestrator and the Controller and also between the Controller and the Network Elements. M.E.L.T. information is stored in local Yang Datastores and accessed by the collector using "pull" mechanisms using the NETCONF (NC), RESTCONF (RC) or gNMI protocols. A "push" strategy is also possible via YANG-Push or gNMI.

1.3. Use Cases

1.3.1. Provisioning root cause analysis

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

With the support for trace context propagation as described in this document for NETCONF, the collector will be able to search every trace, event, metric, or log in connection to that trace-id and faciliate the performance of a root cause analysis due to a network changes. The trace information could also be included as an optional resource with the different NETCONF transaction ids described in [I-D.ietf-netconf-transaction-id].

1.3.2. System performance profiling

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

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

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

1.3.3. Billing and auditing

In certain circumstances, we could envision tracing infomration used as additional inputs to billing systems. In particular, trace context information could be used to validate that a certain northbound order was carried out in southbound systems.

1.4. Terminology

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

The XML prefixes used in this document are mapped as follows:

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

* xmlns:w3ctc="urn:ietf:params:xml:ns:netconf:w3ctc:1.0" and

* xmlns:ietf-netconf-otlp-context=
  "urn:ietf:params:xml:ns:yang:ietf-netconf-otlp-context"

2. NETCONF Extension

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

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

For example, a NETCONF client might send:

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

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

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

For example, a NETCONF client might send:

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

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

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

2.1. Error handling

The NETCONF server SHOULD follow the "Processing Model for Working with Trace Context" as specified in [W3C-Trace-Context].

If the server rejects the RPC because of the trace context extension value, the server MUST return an rpc-error with the following values:

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

Additionally, the error-info tag SHOULD contain a otlp-trace-context-error-info structure with relevant details about the error. This structure is defined in the module ietf-netconf-otlp-context.yang. Example of a badly formated trace context extension:

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

This might give the following error response:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
            xmlns:w3ctc="urn:ietf:params:xml:ns:netconf:w3ctc:1.0"
            xmlns:ietf-netconf-otlp-context=
            "urn:ietf:params:xml:ns:yang:otlp-context"
            message-id="1">
  <rpc-error>
    <error-type>protocol</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-message>
      OTLP traceparent attribute incorrectly formatted
    </error-message>
    <error-info>
      <ietf-netconf-otlp-context:meta-name>
        w3ctc:traceparent
      </ietf-netconf-otlp-context:meta-name>
      <ietf-netconf-otlp-context:meta-value>
        Bad Format
      </ietf-netconf-otlp-context:meta-value>
      <ietf-netconf-otlp-context:error-type>
        ietf-netconf-otlp-context:bad-format
      </ietf-netconf-otlp-context:error-type>
    </error-info>
  </rpc-error>
</rpc-reply>
```

2.2. Trace Context extension versionning

This extension refers to the [W3C-Trace-Context] trace context capability. The W3C traceparent and trace-state headers include the notion of versions. It would be desirable for a NETCONF client to be able to discover the one or multiple versions of these headers supported by a server. We would like to achieve this goal avoiding the deffinition of new NETCONF capabilities for each headers' version.

We define a pair YANG modules (ietf-netconf-otlp-context-traceparent-version-1.0.yang and ietf-netconf-otlp-context-tracestate-version-1.0.yang) that SHOULD be included in the YANG library per [RFC8525] of the NETCONF server supporting the NETCONF Trace Context extension. These capabilities that will refer to the headers' supported versions. Future updates of this document could include additional YANG modules for new headers' versions.

3. YANG Modules

3.1. YANG module for otlp-trace-context-error-info structure

```
<CODE BEGINS>
module ietf-netconf-otlp-context {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:otlp-context";
  prefix ietf-netconf-otlp-context;

  import ietf-yang-structure-ext {
    prefix sx;
    reference "RFC8791: 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>";

  description
    "When propagating tracing information across applications,
    client and servers needs to share some specific contextual
    information. This NETCONF extensions aligns the NETCONF
    protocol to the W3C trace-context document:
    https://www.w3.org/TR/2021/REC-trace-context-1-20211123

    Copyright (c) <year> 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 2023-07-01 {
```

```
    description
      "Initial revision";
    reference
      "RFC XXXX";
  }

  identity meta-error {
    description "Base identity for otlp attribute errors.";
  }

  identity missing {
    base meta-error;
    description "Indicates an attribute or header that is required
      (in the current situation) is missing.";
  }
  identity bad-format {
    base meta-error;
    description "Indicates an attribute or header value where the
      value is incorrectly formatted.";
  }
  identity processing-error {
    base meta-error;
    description "Indicates that the server encountered a processing
      error while processing the attribute or header value.";
  }

  typedef meta-error-type {
    type identityref {
      base meta-error;
    }
    description "Error type";
  }

  sx:structure otlp-trace-context-error-info {
    container otlp-trace-context-error-info {
      description
         "This error is returned by a NETCONF or RESTCONF server
         when a client sends a NETCONF RPC with additonal
         attributes or RESTCONF RPC with additional headers
         for trace context processing, and there is an error
         related to them.  The server has aborted the RPC.";
      leaf meta-name {
        type string;
        description
          "The name of the problematic or missing meta information.
          In NETCONF, the qualified XML attribute name.
          In RESTCONF, the HTTP header name.
          If a client sent a NETCONF RPC with the attribute
```

```
            w3ctc:traceparent='incorrect-format'
            this leaf would have the value 'w3ctc:traceparent'";
        }
        leaf meta-value {
          type string;
          description
            "The value of the problematic meta information received
            by the server.
            If a client sent a NETCONF RPC with the attribute
            w3ctc:traceparent='incorrect-format'
            this leaf would have the value 'incorrect-format'.";
        }
        leaf error-type {
          type meta-error-type;
          description
            "Indicates the type of OTLP meta information problem
            detected by the server.
            If a client sent an RPC annotated with the attribute
            w3ctc:traceparent='incorrect-format'
            this leaf might have the value
            'ietf-netconf-otlp-context:bad-format'.";
        }
      }
    }
  }
  <CODE ENDS>
```

3.2. YANG module for traceparent header version 1.0

```
  <CODE BEGINS>
  module ietf-netconf-otlp-context-traceparent-version-1.0 {
    namespace "urn:ietf:params:xml:ns:yang:traceparent:1.0";
    prefix ietf-netconf-otlpparent-1.0;
  }
  <CODE ENDS>
```

3.3. YANG module for tracestate header version 1.0

```
  <CODE BEGINS>
  module ietf-netconf-otlp-context-tracestate-version-1.0 {
    namespace "urn:ietf:params:xml:ns:yang:tracestate:1.0";
    prefix ietf-netconf-otlpstate-1.0;
  }
  <CODE ENDS>
```

4. Security Considerations

TODO Security

5. IANA Considerations

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

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

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

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

Registrant Contact: The IETF IESG.

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

This document registers three module names in the 'YANG Module Names' registry, defined in RFC 6020:

name: ietf-netconf-otlp-context-traceparent-version-1.0

prefix: ietf-netconf-otlpparent-1.0

namespace: urn:ietf:params:xml:ns:yang:traceparent:1.0

RFC: XXXX

and

name: ietf-netconf-otlp-context-tracestate-version-1.0

prefix: ietf-netconf-otlpstate-1.0

namespace: urn:ietf:params:xml:ns:yang:tracestate:1.0

RFC: XXXX

and

```
     name: ietf-netconf-otlp-context

     prefix: ietf-netconf-otlp-context

     namespace: urn:ietf:params:xml:ns:yang:otlp-context

     RFC: XXXX
```

6. Acknowledgments

TBD

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/rfc/rfc2119>.

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

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

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

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

7.2. Informative References

[I-D.ietf-netconf-transaction-id] Lindblad, J., "Transaction ID Mechanism for NETCONF", Work in Progress, Internet-Draft, draft-ietf-netconf-transaction-id-02, 10 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-transaction-id-02>.

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

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

[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>.

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

Appendix A. Changes (to be deleted by RFC Editor)

A.1. From version 03 to draft-netconf-trace-ctx-extension-00-00

* Adopted by NETCONF WG
* Moved repository to NETCONF WG
* Changed build system to use martinthomson's excellent framework
* Ran make fix-lint to remove white space at EOL etc.
* Added this change note. No other content changes.

A.2. From version 02 to 03

* Changed IANA section to IESG per IANA email
* Created sx:structure and improved error example
* Added ietf-netconf-otlp-context.yang for the sx:structure
* Created a dedicated section for the YANG modules

A.3. From version 01 to 02

* Added Error Handling intial section
* Added how to manage versioning by defining YANG modules for each traceparent and trastate versions as defined by W3C.

* Added 'YANG Module Names' to IANA Considerations

A.4. From version 00 to 01

* Added new section: Implementation example 2: YANG DataStore

* Added new use case: Billing and auditing

* Added in introduction and in "Provisioning root cause analysis" the idea that the different transaction-ids defined in [I-D.ietf-netconf-transaction-id] could be added as part of the tracing information to be exported to the collectors, showing how the two documents are complementary.

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

* Security Considerations

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

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

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

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

XML Attributes Pro:

* Literal alignment with W3C specification

* Same encoding for RESTCONF and NETCONF enabling code reuse

* One specification for all current and future rpcs

XML Attributes Cons:

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

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

RPCs Input Augmentations Pro:

* YANG model of every leaf

* Re-use of YANG toolkits

* Simple updates by augmentations on existing YANG module

* Possibility to express deviations in case of partial support

RPCs Input Augmentations Cons:

* Need to augment every rpc, including future rpcs would need to consider these augmentations, which is harder to maintain

* There is no literal alignment with W3C standard. However, as mentioned before most of the time there will be modifications to the content

* Would need updated RFP for each change at W3C, which will make adoption of new features slower

Authors' Addresses

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

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

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com