NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2024

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

7 July 2023

Subscription to Distributed Notifications
draft-ietf-netconf-distributed-notif-07

Abstract

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

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 January 2024.

Table of Contents

1. Introduction

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

By streaming data from publishers to receivers, much better performance and fine-grained sampling can be achieved than with polling. In a distributed forwarding system, the packet forwarding is delegated to multiple processors on line cards. To not to overwhelm the route processor resources, it is not uncommon that data records are published directly from processors on line cards to target Receivers to further increase efficiency on the routing system.

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

2. Terminologies

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

Subscriber

Publisher

Receiver

Subscription

In addition, this document defines the following terms:

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

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

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

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

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

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

Observation Domain: An Observation Domain is the largest set of observation points for which metrics can be collected, stored and exported by an Agent. For example, a router line card may be an Observation Domain if it is composed of several interfaces, each of which is an observation point.

Observation Domain ID: A 32-bit identifier of the Observation Domain that is locally unique to the network node. In the YANG notification messages it generates, the Observation Domain includes its Observation Domain ID. That way, the collecting process can identify the specific Observation Domain from the publisher that sends the YANG notification messages. Receivers SHOULD use the transport session and the Observation Domain ID field to separate different publisher streams originating from the same publisher.

3. Motivation

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

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

4. Solution Overview

Figure 1 below shows the distributed data export framework.

A collector usually includes two components,

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

* the Receiver is the target for the data publication.

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

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

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

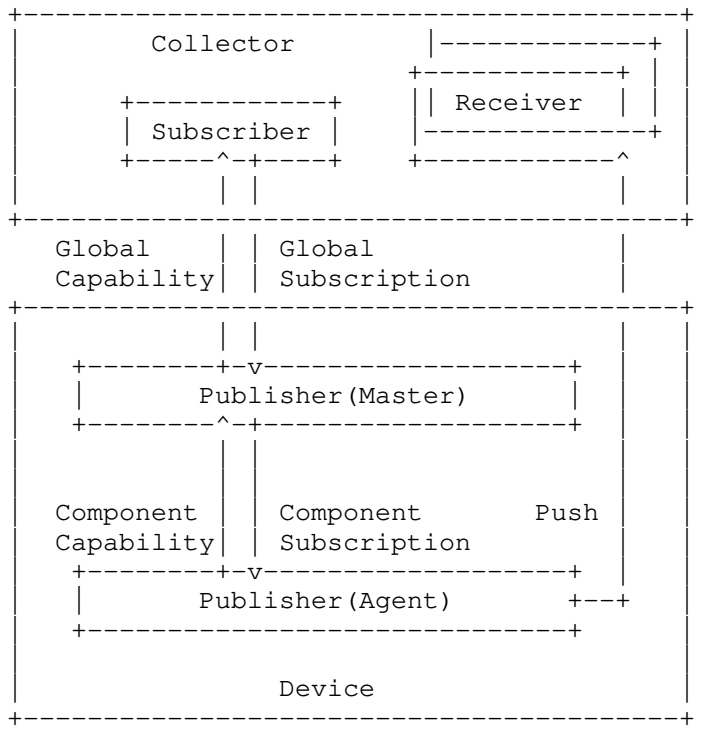


Figure 1: The Distributed Data Export Framework

Master and Agents interact with each other in several ways:

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

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

* The Master relays the component subscriptions to the Agents.

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

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

5. Subscription Decomposition

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

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

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

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

And the Agent to:

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

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

* share the same Subscription ID as the Global Subscription.

6. Publication Composition

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

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

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

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

7. Subscription State Change Notifications

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

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

8. Publisher Configurations

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

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

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

9. YANG Tree

```
module: ietf-distributed-notif

  augment /sn:subscriptions/sn:subscription:
    +--ro message-observation-domain-id*   uint32
  augment /sn:subscription-started:
    +--ro message-observation-domain-id*   uint32
  augment /sn:subscription-modified:
    +--ro message-observation-domain-id*   uint32
  augment /sn:establish-subscription/sn:output:
    +--ro message-observation-domain-id*   uint32
```

10. YANG Module

```
<CODE BEGINS> file "ietf-distributed-notif@2023-03-06.yang"
module ietf-distributed-notif {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-distributed-notif";
  prefix dn;
  import ietf-subscribed-notifications {
    prefix sn;
  }

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

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

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


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

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

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

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

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

     revision 2023-03-06 {
       description
         "Initial version";
       reference
         "RFC XXXX: Subscription to Distributed Notifications";
     }

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

       leaf-list message-observation-domain-id {
         type uint32;
         config false;
         ordered-by user;
         description
           "Software process which created the message (e.g.,
            processor 1 on line card 1). This field is
            used to  notify the collector the working originator.";
       }
     }

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

       uses message-observation-domain-ids;
     }

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

       uses message-observation-domain-ids;
     }

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

```
      uses message-observation-domain-ids;
    }

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

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

11. IANA Considerations

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

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

Registrant Contact: The IESG.

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

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

Name: ietf-distributed-notif

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

Prefix: dn

Reference: RFC XXXX

12. Implementation Status

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

12.1. Open Source Publisher

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

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

12.2. Open Source Receiver Library

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

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

12.3. Pmacct Data Collection

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

12.4. Huawei VRP

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

13. Operational Considerations

In Section 2 of [RFC7011] the Observation Domain term has been used to identify the largest set of Observation Points for which IPFIX Flow information can be aggregated by a Metering Process which usually applies to a router line card. Note that, even if the terms are similar, the concepts are different: indeed, in this case, one router line card can contain multiple publisher processes.

The Observation Domain ID term, issue from IPFIX [RFC7011], has been kept, as opposed to create a new term such as Publisher ID, it is expected that most network node with line cards will contain just one publisher per line card. In which case, the observation domain (ID) concepts in IPFIX and in this draft are similar.

14. Security Considerations

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

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

The new data nodes introduced in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get-config or notification) to this data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

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

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

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

15. Contributors

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

16. Acknowledgements

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

17. References

17.1. Normative References

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

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

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

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

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

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

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

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

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

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

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

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

17.2. Informative References

[I-D.ietf-netconf-https-notif]
Jethanandani, M. and K. Watsen, "An HTTPS-based Transport for YANG Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-https-notif-13, 4 November 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-13>.

[I-D.ietf-netconf-udp-notif]
Zheng, G., Zhou, T., Graf, T., Francois, P., Feng, A. H., and P. Lucente, "UDP-based Transport for Configured Subscriptions", Work in Progress, Internet-Draft, draft-ietf-netconf-udp-notif-09, 10 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-udp-notif-09>.

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

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

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

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

Appendix A. Examples

This appendix is non-normative.

A.1. Dynamic Subscription

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

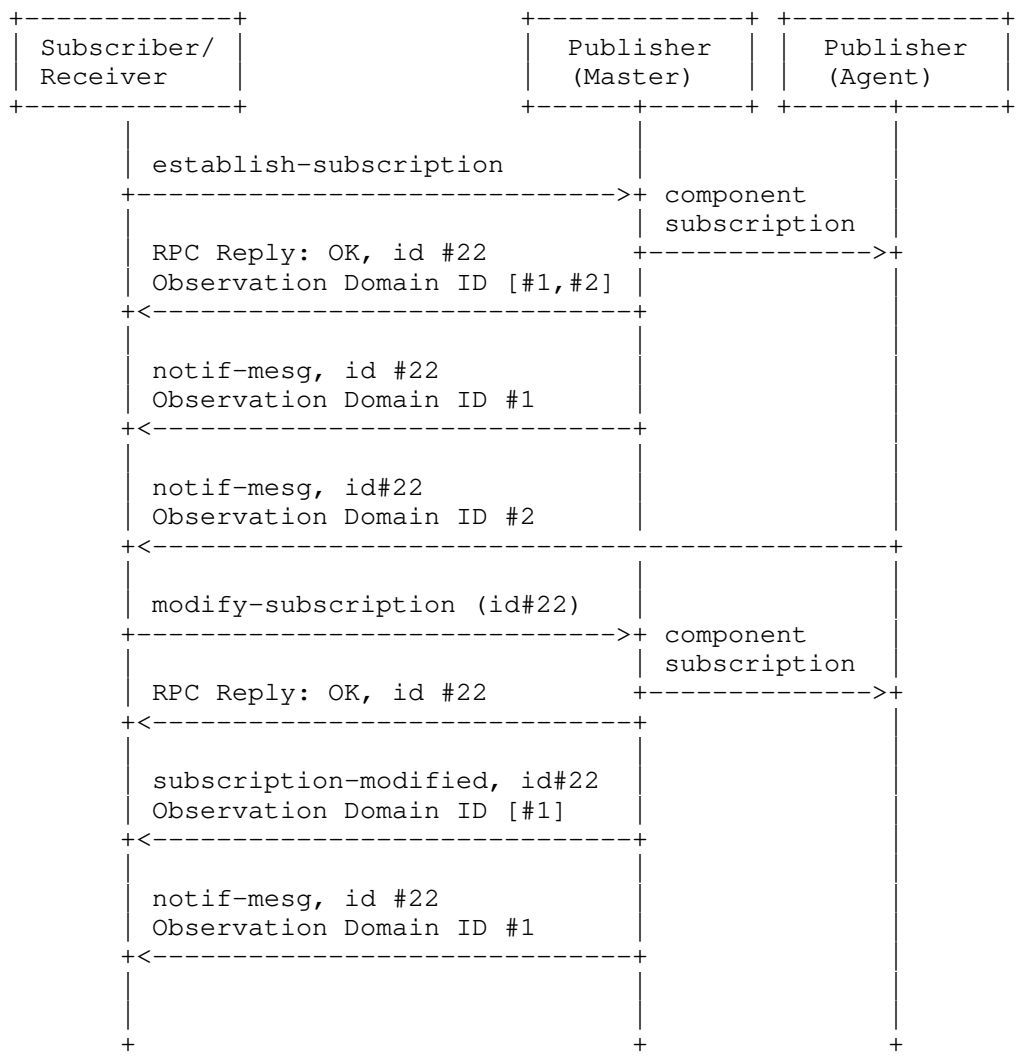


Figure 2: Call Flow for Dynamic Subscription

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

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

Figure 3: "establish-subscription" Request

As the device is able to fully satisfy the request, the request is given a subscription ID of 22. The response as in Figure 4 indicates that the subscription is decomposed into two component subscriptions which will be published by two message Observation Domain ID: #1 and #2.

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

Figure 4: "establish-subscription" Positive RPC Response

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

The subscriber may invoke the "modify-subscription" RPC for a subscription it previously established. The RPC has no difference to the single publisher case as in [RFC8641]. Figure 5 provides an example where a subscriber attempts to modify the period and datastore XPath filter of a subscription using NETCONF.

```
     <rpc message-id="102"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <modify-subscription
        xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
        xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
      <id>22</id>
      <yp:datastore
          xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
        ds:operational
      </yp:datastore>
      <yp:datastore-xpath-filter
          xmlns:ex="https://example.com/sample-data/1.0">
        /ex:bar
      </yp:datastore-xpath-filter>
      <yp:periodic>
        <yp:period>250</yp:period>
      </yp:periodic>
     </modify-subscription>
  </rpc>
```

Figure 5: "modify-subscription" Request

If the modification is successfully accepted, the "subscription-modified" subscription state notification is sent to the subscriber by the Master. The notification, Figure 6 for example, indicates the modified subscription is decomposed into one component subscription which will be published by message Observation Domain #1.

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

Figure 6: "subscription-modified" Subscription State Notification

A.2. Configured Subscription

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

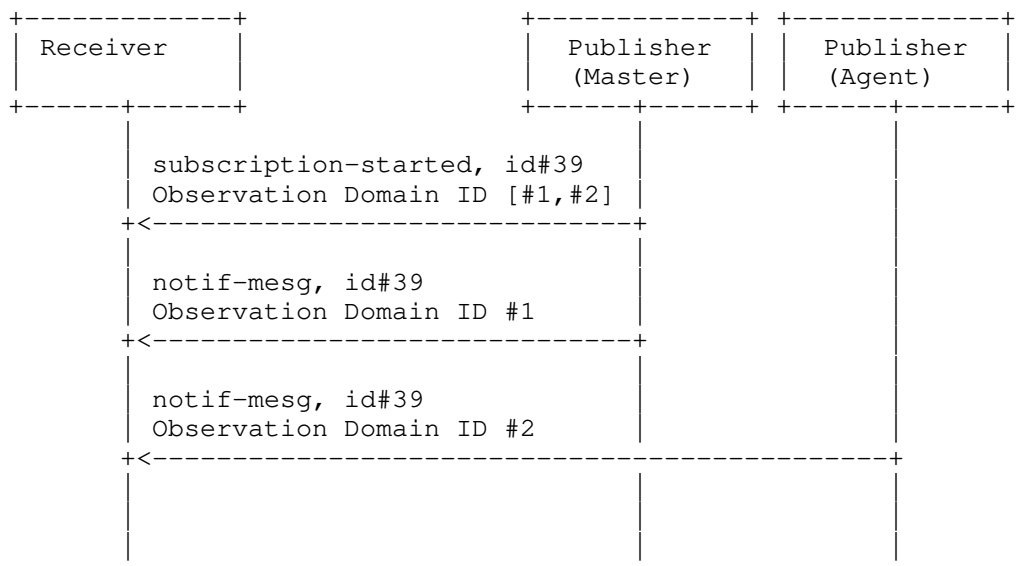


Figure 7: Call Flow for Configured Subscription

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

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

Figure 8: "subscription-started" Subscription State Notification

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

Authors' Addresses

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

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

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

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

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

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 5 January 2024

J. Lindblad
Cisco Systems

4 July 2023

Transaction ID Mechanism for NETCONF
draft-ietf-netconf-transaction-id-01

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 5 January 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

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

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

3.3. Initial Configuration Retrieval

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

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

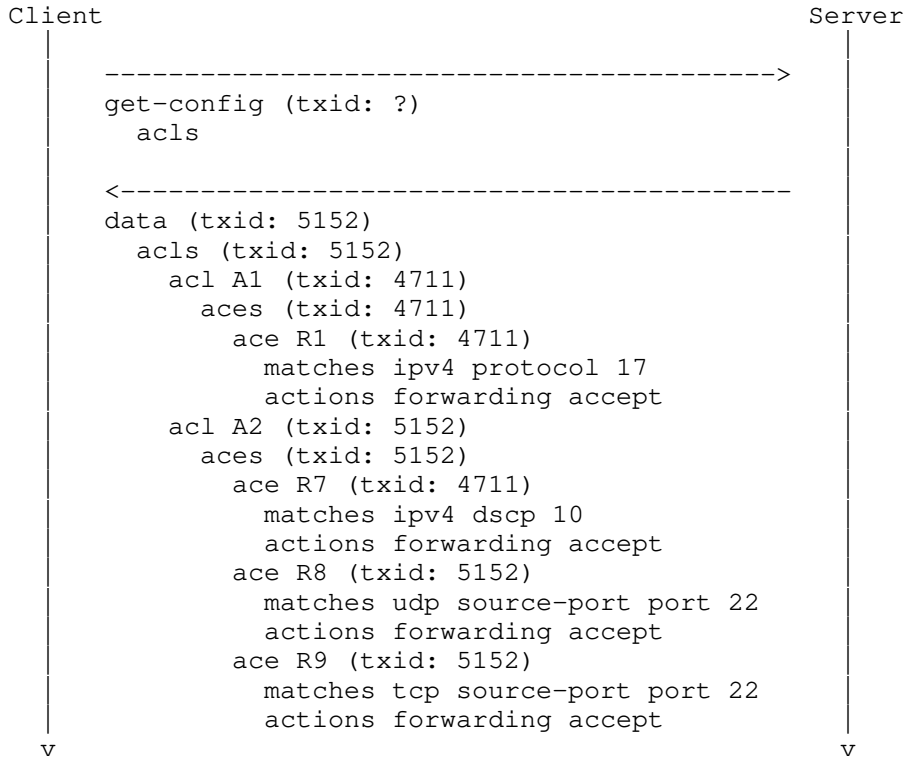


Figure 1: Initial Configuration Retrieval. The client annotated the get-config request itself with the txid request value, which makes the server return all txid values in the entire datastore, that also fall within the requested subtree filter. The most recent change seems to have been an update to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

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 reflect a typical implementation. Servers may assign values randomly. In general, no information can be derived by observing that some txid value is numerically or lexicographically lower than another txid value. In general, the only operation defined on a pair of txid values is testing them for equality.

3.4. Subsequent Configuration Retrieval

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

When a client sends in a txid value of a node that matches the server's txid value for that versioned node, 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, it is obviosuly already up to date with that part of the configuration, so sending it again would be a waste of time and energy.

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

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

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

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

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

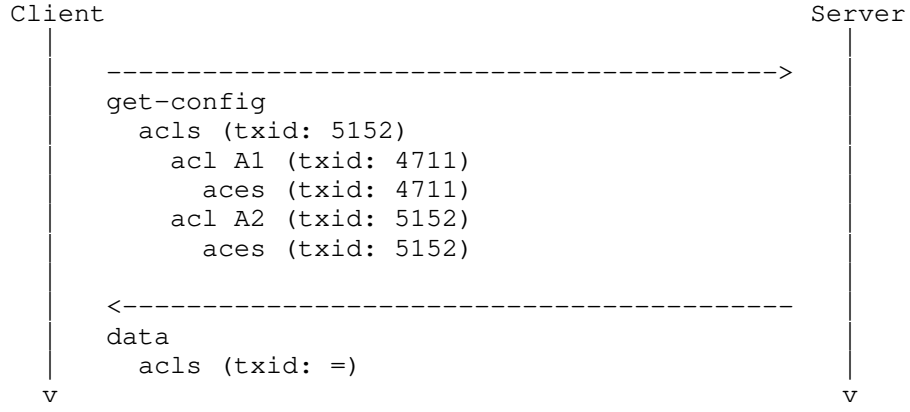


Figure 2: Response Pruning. Client sends get-config request with known txid values. Server prunes response where txid matches expectations.

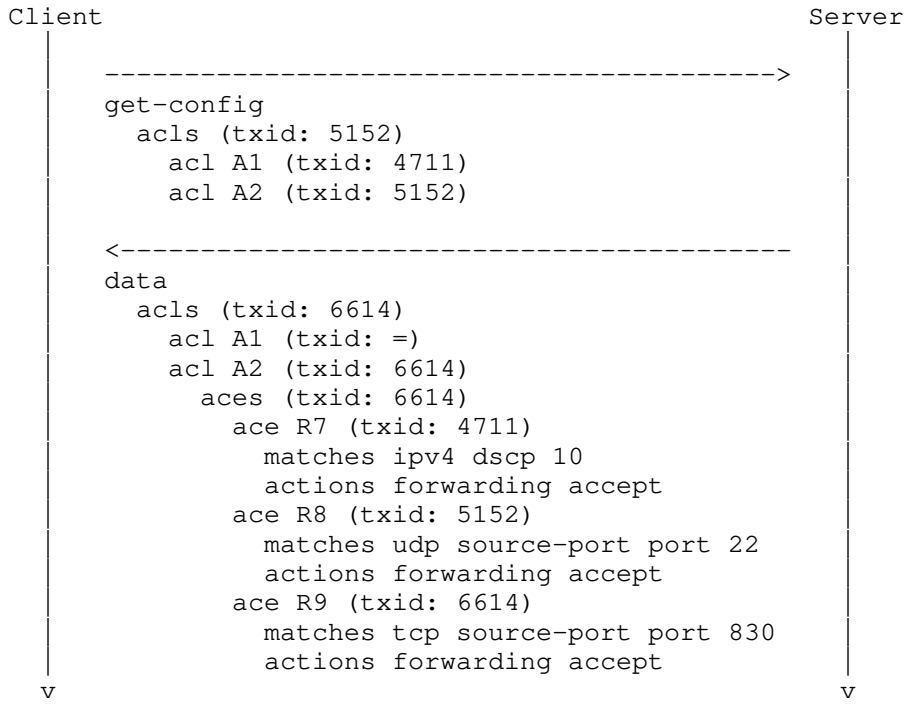


Figure 3: Out of band change detected. Client sends get-config request with known txid values. Server provides update where changes have happened. Specifically ace R8 is returned since ace R8 is a child of a node for which the request had a different txid than the server, and the client did not specify any matching txid for the ace R8 node.

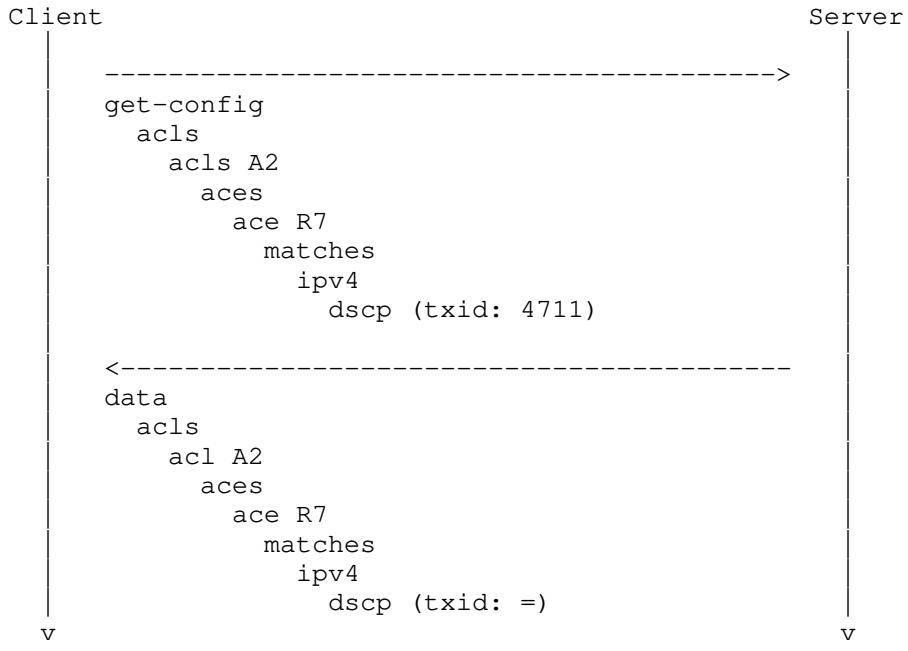


Figure 4: Versioned nodes. Server lookup of dscp txid gives 4711, as closest ancestor is ace R7 with txid 4711. Since the server's and client's txid match, the etag value is '=', and the leaf value is pruned.

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 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 txid value the node will have after commit, assuming the client makes no further changes of the candidate datastore.

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

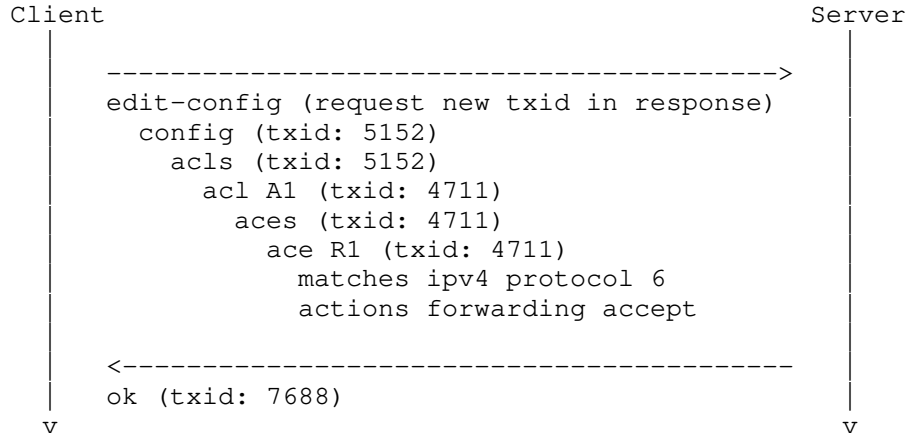


Figure 5: Conditional transaction towards the Running datastore successfully executed. As all the txid values specified by the client matched those on the server, the transaction was successfully executed.

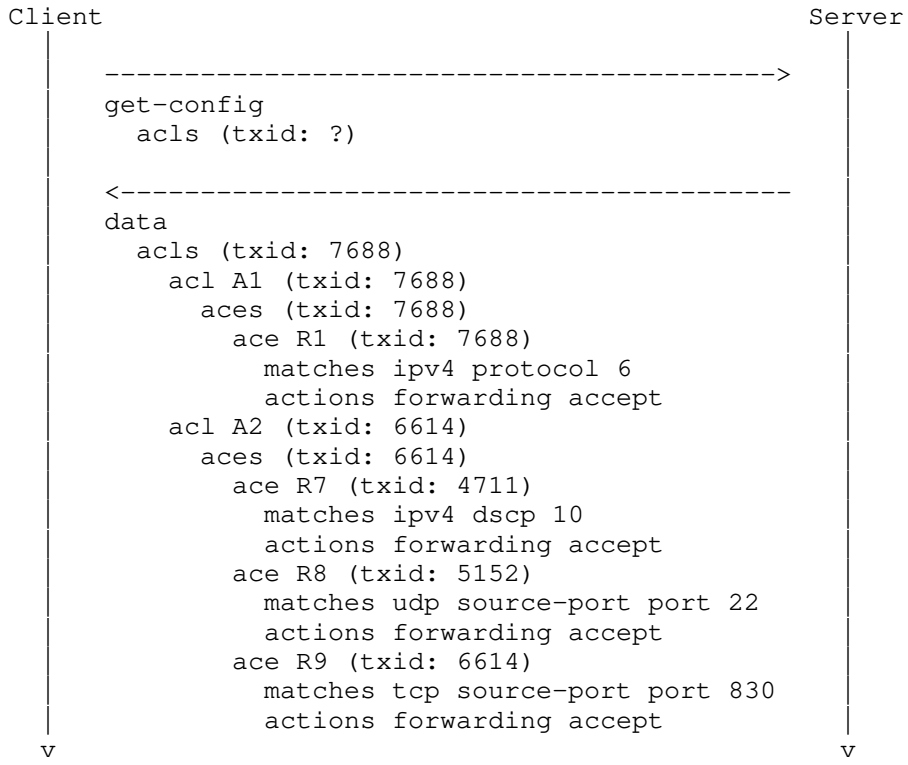


Figure 6: The txids are updated on all versioned nodes that were modified themselves or have a child node that was modified.

If the server rejects the transaction because one or more of the configuration 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.

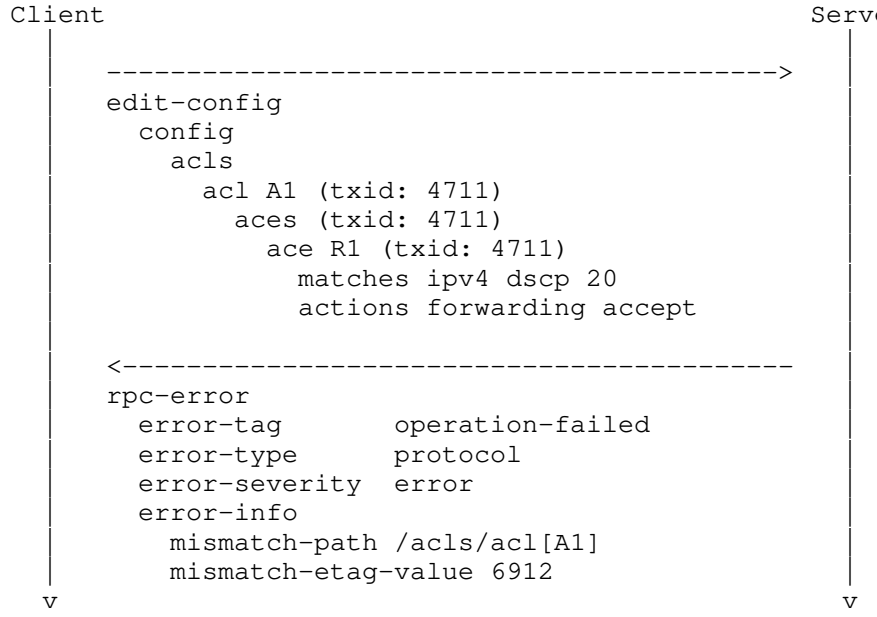
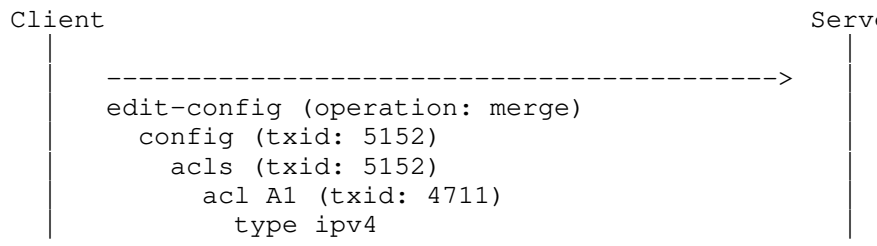


Figure 7: Conditional transaction that fails a txid check. The client wishes to ensure there has been no changes to the particular acl entry it edits, and therefore sends the txid it knows for this part of the configuration. Since the txid has changed (out of band), the server rejects the configuration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

3.7. 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 txid attributes to the configuration payload the same way. In case a client specifies different txid values for the same element in successive edit-config or edit-data operations, the txid value specified last MUST be used by the server at commit time.



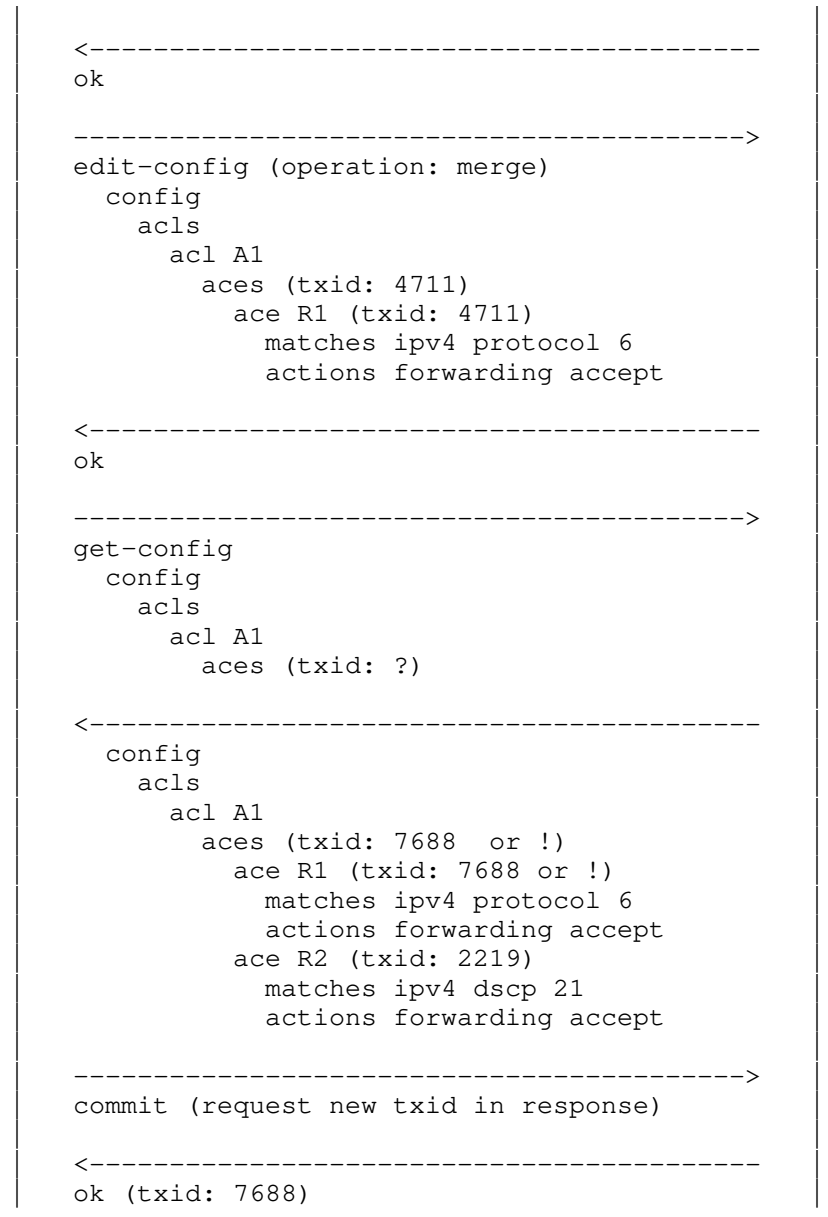

<----------
ok
---------->
edit-config (operation: merge)
config
acls
acl A1
aces (txid: 4711)
ace R1 (txid: 4711)
matches ipv4 protocol 6
actions forwarding accept
<----------
ok
---------->
get-config
config
acls
acl A1
aces (txid: ?)
<----------
config
acls
acl A1
aces (txid: 7688 or !)
ace R1 (txid: 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)

Figure 8: Conditional transaction towards the Candidate datastore successfully executed. As all the 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 txid value that would be used if the candidate was committed without further changes (when that 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 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:

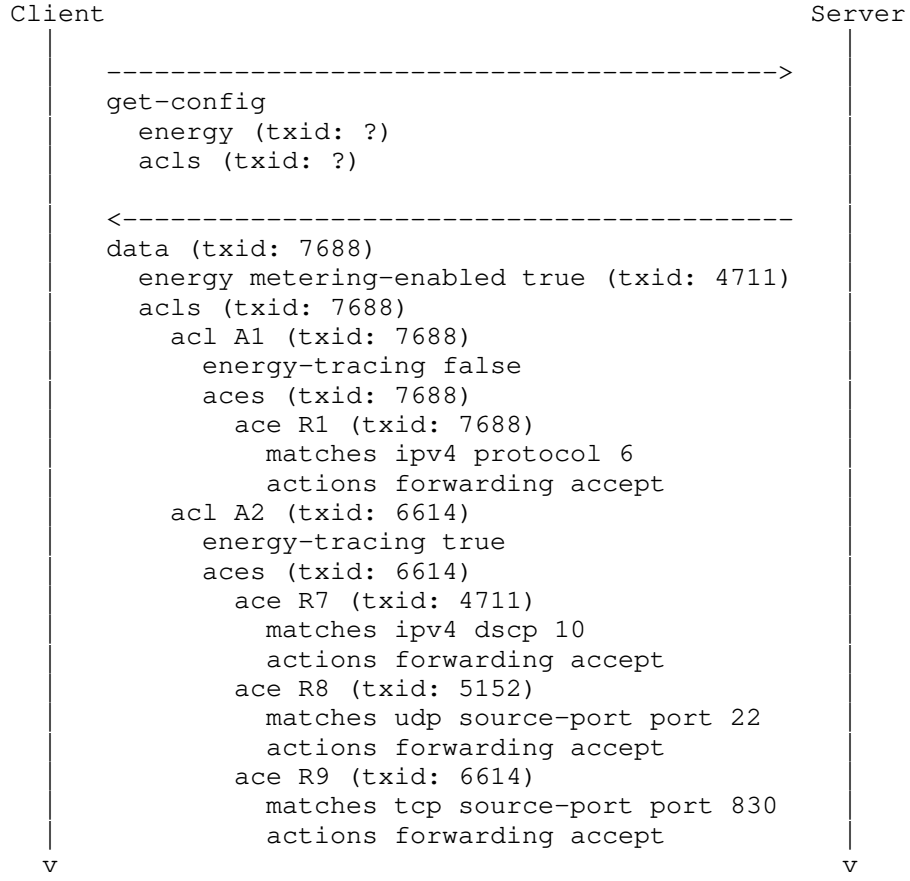


Figure 9: Initial configuration for the energy example. Note the energy metering-enabled leaf at the top and energy-tracing leafs under each acl.

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

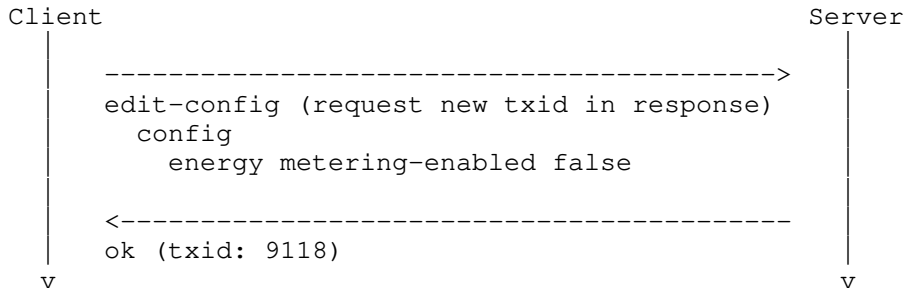


Figure 10: Transaction changing a single leaf. This leaf is the target of a when-statement, however, which means other leafs elsewhere may be indirectly modified by this change. Such indirect changes will also result in txid changes.

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

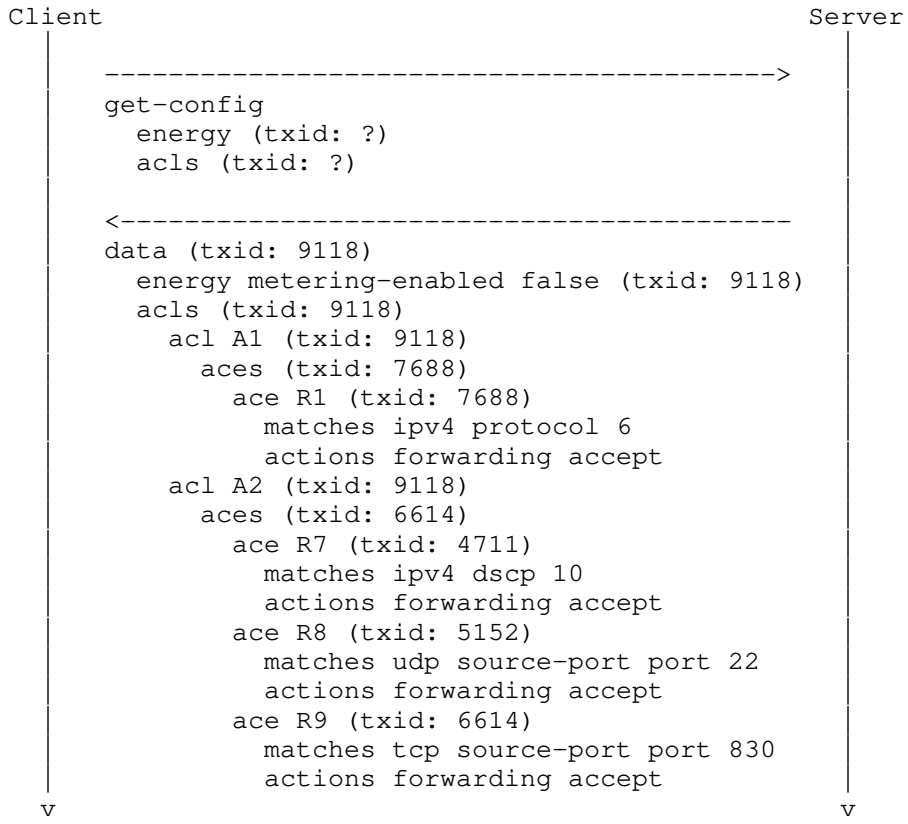


Figure 11: The txid for the energy subtree has changed since that was the target of the edit-config. The txids of the ACLs have also changed since the energy-tracing leafs are now removed by the now false when- expression. Both acl A1 and acl A2 have their txids updated, even though energy-tracing was 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).

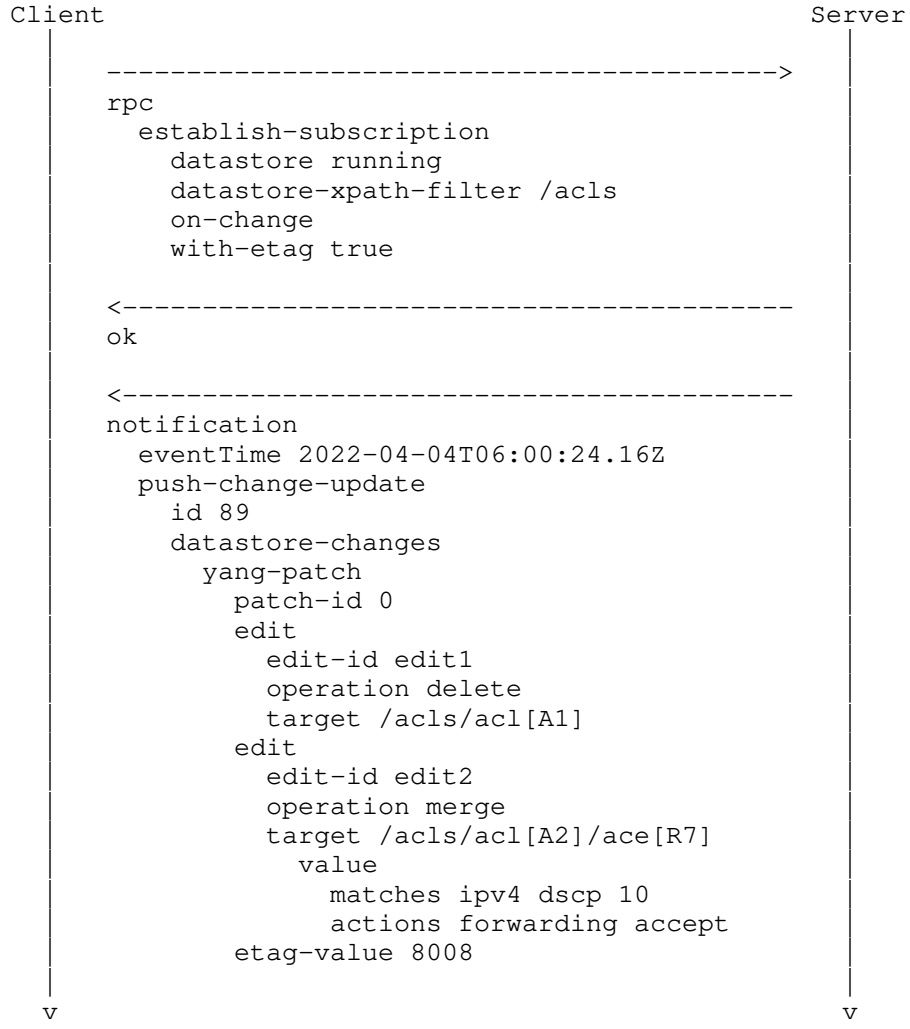


Figure 12: A client requests a YANG-Push subscription for a given path with txid value included. When the server delivers a push-change-update notification, the txid value pertaining to the entire patch is included.

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.

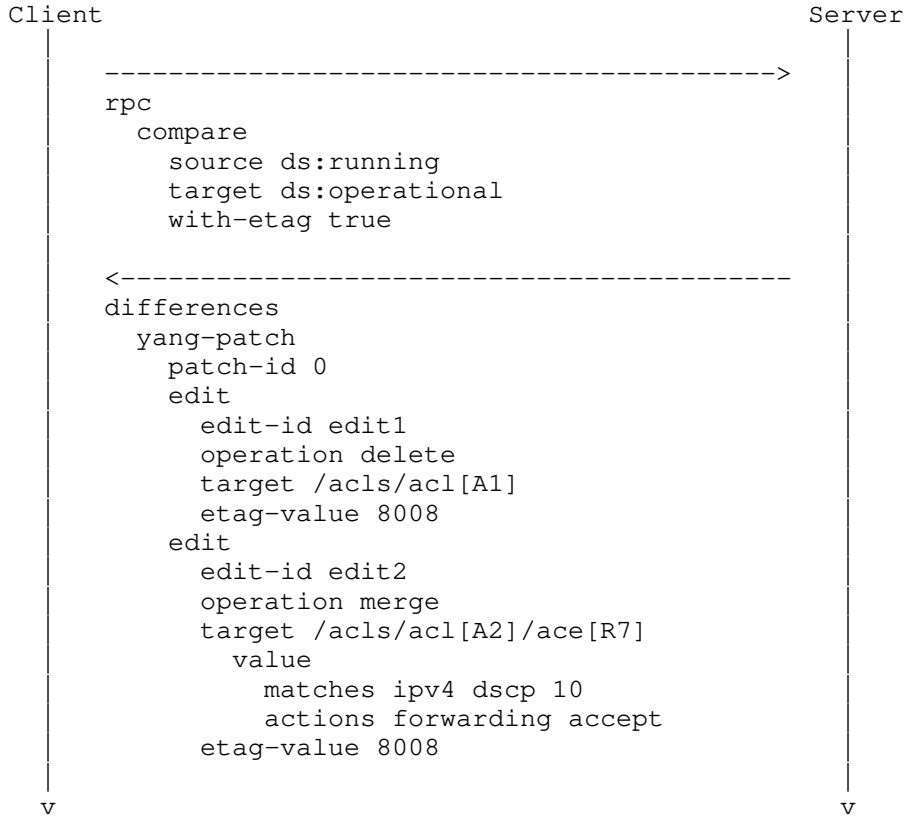


Figure 13: A client requests a NMDA Datastore compare for a given path with txid values included. When the server delivers the reply, the txid is included for each edit.

4. Txid Mechanisms

This document defines two txid mechanisms:

* The etag attribute txid mechanism

* The last-modified attribute txid mechanism

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

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

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

4.1. The etag attribute txid mechanism

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

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

The etag attribute values are opaque UTF-8 strings chosen freely, except that the etag string must not contain space, backslash or double quotes. The point of this restriction is to make it easy to reuse implementations that adhere to section 2.3.1 in [RFC7232]. The probability SHOULD be made very low that an etag value that has been used historically by a server is used again by that server if the configuration is different.

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

The detailed rules for when to update the etag value are described in section General Txid Principles (Section 3.2). These rules are chosen to be consistent with the ETag mechanism in RESTCONF, [RFC8040], specifically sections 3.4.1.2, 3.4.1.3 and 3.5.2.

4.2. The last-modified attribute txid mechanism

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

NETCONF servers that support this extension MUST announce the feature txid-last-modified defined in ietf-netconf-txid.yang.

The last-modified attribute values are yang:date-and-time values as defined in ietf-yang-types.yang, [RFC6991].

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

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

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

RESTCONF, as defined in [RFC8040], is using a different format for the time stamps which is limited to one second resolution. Server implementors that support the Last-Modified txid mechanism over both RESTCONF and other management protocols are RECOMMENDED to use Last-Modified timestamps that match the point in time referenced over RESTCONF, with the fractional seconds part added.

The detailed rules for when to update the last-modified value are described in section General Txid Principles (Section 3.2). These rules are chosen to be consistent with the Last-Modified mechanism in RESTCONF, [RFC8040], specifically sections 3.4.1.1, 3.4.1.3 and 3.5.1.

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

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

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

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

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

Clients MAY request servers that also implement YANG-Push to return configuration change subsription updates with etag or last-modified txid attributes. The client requests this service by adding a with-etag or with-last-modified flag with the value 'true' to the subscription request or yang-push configuration. The server MUST then return such txids on the YANG Patch edit tag and to the child elements of the value tag. The txid attribute on the edit tag reflects the txid associated with the changes encoded in this edit section, as well as parent nodes. Later edit sections in the same push-update or push-change-update may still supercede the txid value for some or all of the nodes in the current edit section.

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

Servers returning a txid mismatch error MUST return an rpc-error as defined in section Conditional Transactions (Section 3.6) with an error-info tag containing a txid-value-mismatch-error-info structure.

4.3.1. Candidate Datastore

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

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

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

4.3.2. Namespaces and Attribute Placement

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

In client messages sent to a server:

* /nc:rpc/nc:get-config

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

* /nc:rpc/ncds:get-data

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

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

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

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

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

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

In server messages sent to a client:

* /nc:rpc-reply/nc:data

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

* /nc:rpc-reply/ncds:data

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

* /nc:rpc-reply/nc:ok

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

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

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

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

5. Txid Mechanism Examples

5.1. Initial Configuration Response

5.1.1. With etag

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

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

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

The server's reply might then be:

```
<rpc-reply message-id="1"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data txid:etag="nc5152">
    <acls xmlns=
            "urn:ietf:params:xml:ns:yang:ietf-access-control-list"
          txid:etag="nc5152">
      <acl txid:etag="nc4711">
        <name>A1</name>
```

```
          <aces txid:etag="nc4711">
            <ace txid:etag="nc4711">
              <name>R1</name>
              <matches>
                <ipv4>
                  <protocol>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, 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>
      <runnign/>
    </target>
    <test-option>test-then-set</test-option>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
    <config>
      <acls xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
        <acl nc:operation="delete"
             txid:etag="nc7688">
          <name>A1</name>
        </acl>
      </acls>
    </config>
  </edit-config>
</rpc>
```

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

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

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

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

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

5.5. Reading from the Candidate Datastore

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

```
<rpc-reply message-id="12"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list"
      txid:etag="nc4711">
      <acl txid:etag="nc4711">
        <name>A1</name>
        <aces txid:etag="nc4711">
          <ace txid:etag="nc4711">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>17</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc2219">
            <name>R2</name>
            <matches>
              <ipv4>
                <dscp>21</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

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

```
<rpc-reply message-id="13"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <data>
    <acls
      xmlns="urn:ietf:params:xml:ns:yang:ietf-access-control-list">
      <acl txid:etag="!">
        <name>A1</name>
        <aces txid:etag="!">
          <ace txid:etag="!">
            <name>R1</name>
            <matches>
              <ipv4>
                <protocol>6</protocol>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
          <ace txid:etag="nc2219">
            <name>R2</name>
            <matches>
              <ipv4>
                <dscp>21</dscp>
              </ipv4>
            </matches>
            <actions>
              <forwarding xmlns:acl=
              "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
                acl:accept
              <forwarding>
            </actions>
          </ace>
        </aces>
      </acl>
    </acls>
  </data>
</rpc-reply>
```

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

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

5.6. Commit

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

For example, a client might send:

```
<rpc message-id="14"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    xmlns:ietf-netconf-txid=
      "urn:ietf:params:xml:ns:yang:ietf-netconf-txid"
  <commit>
    <ietf-netconf-txid:with-etag>true<ietf-netconf-txid:with-etag>
  </commit>
</rpc>
```

Assuming the server accepted the transaction, it might respond:

```
<rpc-reply message-id="15"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:txid="urn:ietf:params:xml:ns:netconf:txid:1.0">
  <ok txid:etag="nc8008"/>
</rpc-reply>
```

5.7. YANG-Push

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

```
<netconf:rpc message-id="16"
             xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
      xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
      xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push"
      xmlns:ietf-netconf-txid-yp=
        "urn:ietf:params:xml:ns:yang:ietf-txid-yang-push">
    <yp:datastore
        xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:running
    </yp:datastore>
    <yp:datastore-xpath-filter
        xmlns:acl=
          "urn:ietf:params:xml:ns:yang:ietf-access-control-list">
      /acl:acls
    </yp:datastore-xpath-filter>
    <yp:on-change/>
    <ietf-netconf-txid-yp:with-etag>
      true
    </ietf-netconf-txid-yp:with-etag>
  </establish-subscription>
</netconf:rpc>
```

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

```
<rpc message-id="17"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
      xmlns=
        "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
      xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push"
      xmlns:ietf-netconf-txid-yp=
        "urn:ietf:params:xml:ns:yang:ietf-txid-yang-push">
    <id>1011</id>
    <yp:datastore
        xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:running
    </yp:datastore>
    <ietf-netconf-txid-yp:with-etag>
      false
    </ietf-netconf-txid-yp:with-etag>
  </modify-subscription>
</rpc>
```

A server might send a subscription update like this:

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

5.8. NMDA Compare

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

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

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

RPC reply when a difference is detected:

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

The same response in RESTCONF (using JSON format):

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

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

6. YANG Modules

6.1. Base module for txid in NETCONF

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

  import ietf-netconf {
    prefix nc;
  }

  import ietf-netconf-nmda {
    prefix ncds;
  }

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

  import ietf-yang-types {
    prefix yang;
  }

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

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

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

  description
    "NETCONF Transaction ID aware operations for NMDA.

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

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

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

```
        for full legal notices.

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

    revision 2023-03-01 {
      description
        "Initial revision";
      reference
        "RFC XXXX: Xxxxxxxxx";
    }

    feature txid-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.";
    }

    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 txid-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 txid-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 txid-last-modified;
          type last-modified-t;
          description
            "Indicates server's txid value of the last-modified
            attribute for one mismatching element.";
        }
      }
    }
  }
  <CODE ENDS>
```

6.2. Additional support for txid in YANG-Push

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

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

    import ietf-yang-push {
      prefix yp;
      reference
        "RFC 8641: Subscriptions to YANG Datastores";
    }

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

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

```
  }

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

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

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

  description
    "NETCONF Transaction ID aware operations for YANG Push.

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

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

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

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

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

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

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

6.3. Additional support for txid in NMDA Compare

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

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

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

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

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

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

     description
       "NETCONF Transaction ID aware operations for NMDA Compare.

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

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

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

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

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

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

```
        txid-values.";
      container most-recent {
        uses ietf-netconf-txid:txid-value-grouping;
      }
    }
  }
  <CODE ENDS>
```

7. Security Considerations

7.1. NACM Access Control

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

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

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

7.1.1. Hash-based Txid Algorithms

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

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

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

7.2. Unchanged Configuration

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

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

8. IANA Considerations

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

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

This document registers four XML namespace URNs in the 'IETF XML registry', following the format defined in [RFC3688].

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

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

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

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

Registrant Contact: The NETCONF WG of the IETF.

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

This document registers three module names in the 'YANG Module Names' registry, defined in [RFC6020].

name: ietf-netconf-txid

prefix: ietf-netconf-txid

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

RFC: XXXX

and

```
     name: ietf-netconf-txid-yp

     prefix: ietf-netconf-txid-yp

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

     RFC: XXXX
```

and

```
     name: ietf-netconf-txid-nmda-compare

     prefix: ietf-netconf-txid-nmda-compare

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

     RFC: XXXX
```

9. Changes

9.1. Major changes in -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.

9.2. 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

9.3. 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.

9.4. Major changes in -01 since -00

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

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

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

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

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

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

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

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

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

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

* Added IANA Considerations section.

* Removed all comments about open questions.

10. References

10.1. Normative References

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

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

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

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

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

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

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

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

10.2. Informative References

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

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

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

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

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

Acknowledgments

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

Author's Address

Jan Lindblad
Cisco Systems
Email: jlindbla@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2024

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

7 July 2023

UDP-based Transport for Configured Subscriptions
draft-ietf-netconf-udp-notif-10

Abstract

This document describes a UDP-based protocol for YANG notifications to collect data from networking devices. 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 8 January 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 devices 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 networking devices.

* 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 device. 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.

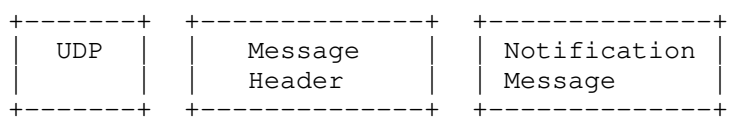


Figure 1: UDP-Notif Message Overview

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.

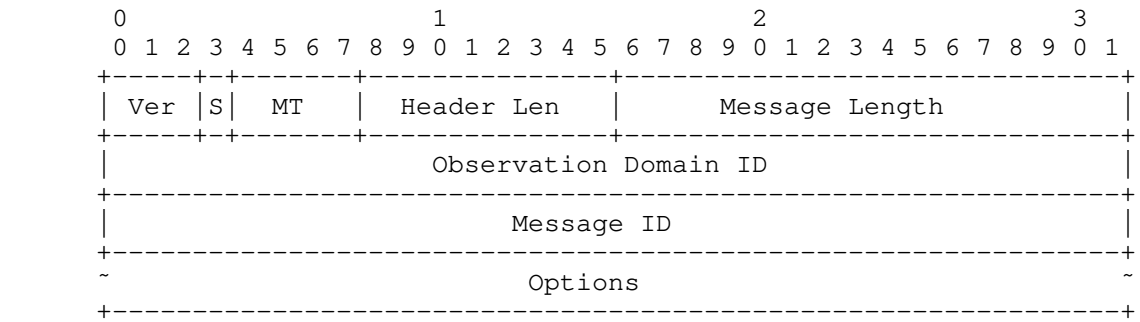


Figure 2: UDP-Notif Message Header Format

The Message Header contains the following field:

* Ver represents the PDU (Protocol Data Unit) encoding version. The current version value is 1.

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

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

  - 0: Reserved;

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

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

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

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

* Message Length is the total length of the message within one UDP datagram, measured in octets, including the message header.

* Observation Domain ID is a 32-bit identifier defined in [I-D.ietf-netconf-distributed-notif]. This identifier is unique within the publisher node and identifies the exporter process of the node allowing the disambiguation of an information source. Message unicity is obtained from the conjonction of the

Observation Domain ID and the Message ID field described below. If Observation Domain ID unicity is not preserved through the collection domain, the source IP address of the UDP datagram SHOULD be used in addition to the Observation Domain ID to identify the information source. If a transport layer relay is used, Observation Domain 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 Observation Domain ID. Note that the main purpose of the Message ID is to reconstruct messages which were segmented using the segmentation option described in section Section 4.1. The Message ID values SHOULD be incremented by one for each successive message originated with the same Observation Domain ID, so that message loss can be detected. Furthermore, incrementing the Message ID by one allows for a large amount of time to happen before the Message ID's are reused due to wrapping around. Different subscribers MAY share the same Message ID sequence.

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

3.3. Data Encoding

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

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

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

4. Options

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

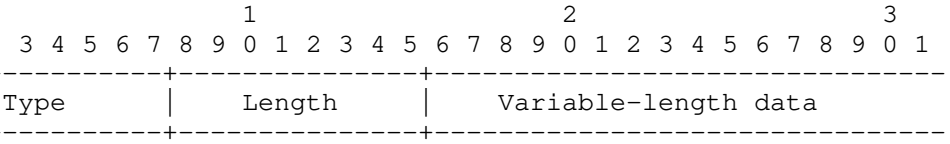


Figure 3: Generic Option Format

* Type: 1 octet describing the option type;

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

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

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

4.1. Segmentation Option

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

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

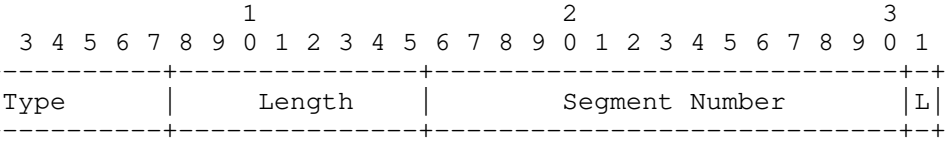


Figure 4: Segmentation Option Format

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

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

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

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

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

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

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

4.2. Private Encoding Option

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

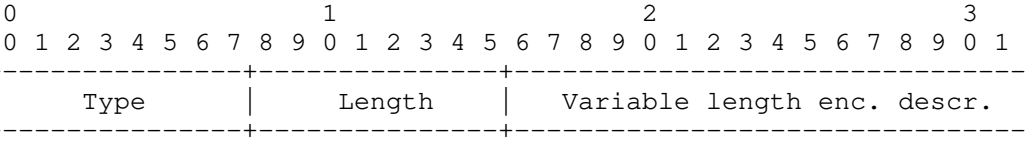


Figure 5: Private Encoding Option Format

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

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

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

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

5. Applicability

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

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

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 device to use a lower streaming rate. The interaction to control the streaming rate on the device is out of the scope of this document.

5.2. Message Size

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

5.3. Reliability

The target application for UDP-Notif is the collection of data-plane information. The lack of reliability of the data streaming mechanism is thus considered acceptable as the mechanism is to be used in controlled environments, mitigating the risk of information loss, while allowing for publication of very large amounts of data. Moreover, in this context, sporadic events when incomplete data collection is provided is not critical for the proper management of the network, as information collected for the devices through the means of the proposed mechanism is to be often refreshed.

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

6. Secured layer for UDP-notif

In open or unsecured networks, UDP-notif messages MUST be secured or encrypted. In this section, a mechanism using DTLS 1.3 to secure UDP-notif protocol is presented. The following sections defines the requirements for the implementation of the secured layer of DTLS for UDP-notif. No DTLS 1.3 extensions are defined nor needed.

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] without any new extensions.

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. When these countermeasures are used, the Receiver responds with a DTLS HelloRetryRequest containing a stateless cookie. The Publisher MUST send a new DTLS ClientHello message containing the received cookie, which initiates the DTLS handshake.

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 privacy is REQUIRED, then implementations must either negotiate a cipher suite that employs a non-NULL encryption algorithm or otherwise achieve privacy by other means, such as a physically secured network.

6.1.2. Publish Data

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

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

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

MSG-LEN = NONZERO-DIGIT *DIGIT

SP = %d32

NONZERO-DIGIT = %d49-57

DIGIT = %d48 / NONZERO-DIGIT
```

UDP-NOTIF-MSG is defined in Section 3.

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

6.1.3. Session termination

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

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

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

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

7.1. Generic grouping for UDP-based applications

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

7.1.1. YANG Tree

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

```
module: ietf-udp-client

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

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

7.1.2. YANG Module

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

<CODE BEGINS> file "ietf-udp-client@2023-05-08.yang"

```
module ietf-udp-client {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-udp-client";
  prefix udpc;
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-tls-client {
    prefix tlsc;
    reference
      "RFC TTTT: YANG Groupings for TLS Clients and TLS Servers";
  }

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

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

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

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

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

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

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

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

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

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

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

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

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

7.2. YANG to configure UDP-notif

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

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

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

7.3. YANG Module

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

<CODE BEGINS> file "ietf-udp-notif-transport@2023-05-08.yang"

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

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

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

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

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

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

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

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

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

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

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

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

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

    uses udpc:udp-client-grouping;

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

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

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

```
}
<CODE ENDS>
```

8. IANA Considerations

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

8.1. IANA registries

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

The first requested registry is the following:

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

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

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

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

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

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

The second requested registry is the following:

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

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

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

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

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

The third requested registry is the following:

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

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

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

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

8.2. URI

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

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

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

8.3. YANG module name

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

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

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

9. Implementation Status

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

9.1. Open Source Publisher

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

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

9.2. Open Source Receiver Library

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

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

9.3. Pmacct Data Collection

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

9.4. Huawei VRP

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

10. Security Considerations

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

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

11. Acknowledgements

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

12. References

12.1. Normative References

[I-D.ietf-netconf-distributed-notif]
Zhou, T., Zheng, G., Voit, E., Graf, T., and P. Francois, "Subscription to Distributed Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-distributed-notif-06, 11 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-distributed-notif-06>.

[I-D.ietf-netconf-https-notif]
Jethanandani, M. and K. Watsen, "An HTTPS-based Transport for YANG Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-https-notif-13, 4 November 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-13>.

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

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

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

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

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

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

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

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

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

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

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

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

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

[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-01, 3 March 2023, <https://datatracker.ietf.org/doc/html/draft-ahuang-netconf-notif-yang-01>.

[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>.

[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>.

[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>.

Appendix A. UDP-notif Examples

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 6: JSON Payload to be sent

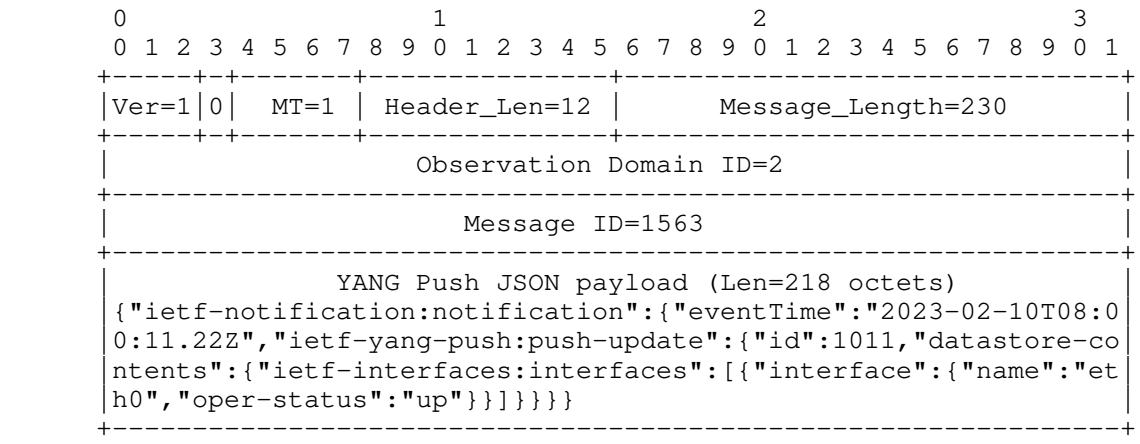


Figure 7: UDP-notif transport message

Authors' Addresses

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

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

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

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

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

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 7 January 2024

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

6 July 2023

Support of Versioning in YANG Notifications Subscription
draft-ietf-netconf-yang-notifications-versioning-01

Abstract

This document extends the YANG notifications subscription mechanism to specify the YANG module semantic version at the subscription. Then, a new extension with the revision and the semantic version of the YANG push subscription state change notification is proposed.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 7 January 2024.

Copyright Notice

Table of Contents

1. Introduction

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

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

This semantic reference is available when the subscription is being establish as described in Section 3.6 of [RFC8641] and being streamed from the publisher to receiver with the subscription state change notifications described in Section 2.7 of [RFC8639] where for each subscription a locally unique subscription ID described in Section 4.3.2 of [RFC8641] is being issued and streamed as metadata with the notification message in the YANG push message header.

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

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

This documents extends the current YANG notifications subscription mechanism to allow to subscribe to a specific revision or latest YANG module semantic version to which the YANG module version needs to be backward compatible to. The subscription state change "subscription-started" and "subscription-modified" notification messages are also extended to include the revision and semantic version.

2. Extend the Datastore Selection

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

module: Restricts the subscription to one or more YANG module names with revision and revision-label together for the related streamed content.

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

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

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

3. Extend the Subscription State Change Notifications

Besides the Subscription ID and the xpath or sub-tree filter reference as described in Section 2.7 of [RFC8639], the following metadata objects are part of a "subscription-started" or "subscription-modified" subscription state change notification.

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

revision: Describes the YANG module revision as specified in Section 7.1.9 of [RFC6020] for the related streamed content.

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

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

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2023-01-03T10:00:00Z</eventTime>
  <subscription-modified
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <id>101</id>
    <stream-xpath-filter
      xmlns:int="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      /int:interfaces
    </stream-xpath-filter>
    <stream>NETCONF</stream>
    <module-version
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push-revision">
      <module-name>ietf-interfaces</module-name>
      <revision>2014-05-08</revision>
      <revision-label>1.0.0</revision-label>
    </module-version>
  </subscription-modified>
</notification>
```

Figure 1: XML Push Example for a subscription-modified notification message

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

```
{
  "ietf-restconf:notification" : {
    "eventTime": "2023-01-03T10:00:00Z",
    "ietf-subscribed-notifications:subscription-modified": {
      "id": 101,
      "stream-xpath-filter": "/ietf-interfaces:interfaces",
      "stream": "NETCONF",
      "ietf-yang-push-revision:module-version": [{
        "module-name": "ietf-interfaces"
        "revision": "2014-05-08",
        "revision-label": "1.0.0",
      }],
    }
  }
}
```

Figure 2: JSON Push Example for a subscription-modified notification message

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

4.1. Data Model Overview

This YANG module augments the "ietf-yang-push" module with the module-name, revision and revision-label in the "subscription-started" and "subscription-modified" subscription state change notifications and the ability to define the "revision" and "revision-label" in the "establish-subscription" and "modify-subscription" RPCs in the datastore push subscription.

4.1.1. Tree View

The following is the YANG tree diagram [RFC8340] for the ietf-yang-push-revision YANG module

```
module: ietf-yang-push-revision

  augment /sn:filters/sn:stream-filter:
    +--rw module-version* [module-name]
       +--rw module-name       yang:yang-identifier
       +--rw revision?         rev:revision-date-or-label
       +--rw revision-label?   ysver:version
  augment /sn:filters/yp:selection-filter:
```

```
      +--rw module-version* [module-name]
         +--rw module-name       yang:yang-identifier
         +--rw revision?         rev:revision-date-or-label
         +--rw revision-label?   ysver:version
    augment /sn:establish-subscription/sn:input/sn:target/sn:stream
              /sn:stream-filter/sn:within-subscription:
      +-- module-version* [module-name]
         +-- module-name       yang:yang-identifier
         +-- revision?         rev:revision-date-or-label
         +-- revision-label?   ysver:version
    augment /sn:establish-subscription/sn:input/sn:target/yp:datastore
              /yp:selection-filter/yp:within-subscription:
      +-- module-version* [module-name]
         +-- module-name       yang:yang-identifier
         +-- revision?         rev:revision-date-or-label
         +-- revision-label?   ysver:version
    augment /sn:modify-subscription/sn:input/sn:target/sn:stream
              /sn:stream-filter/sn:within-subscription:
      +-- module-version* [module-name]
         +-- module-name       yang:yang-identifier
         +-- revision?         rev:revision-date-or-label
         +-- revision-label?   ysver:version
    augment /sn:modify-subscription/sn:input/sn:target/yp:datastore
              /yp:selection-filter/yp:within-subscription:
      +-- module-version* [module-name]
         +-- module-name       yang:yang-identifier
         +-- revision?         rev:revision-date-or-label
         +-- revision-label?   ysver:version
    augment /sn:subscription-started/sn:target/sn:stream
              /sn:stream-filter/sn:within-subscription:
      +--ro module-version* [module-name]
         +--ro module-name       yang:yang-identifier
         +--ro revision          rev:revision-date-or-label
         +--ro revision-label?   ysver:version
    augment /sn:subscription-started/sn:target/yp:datastore
              /yp:selection-filter/yp:within-subscription:
      +--ro module-version* [module-name]
         +--ro module-name       yang:yang-identifier
         +--ro revision          rev:revision-date-or-label
         +--ro revision-label?   ysver:version
    augment /sn:subscription-modified/sn:target/sn:stream
              /sn:stream-filter/sn:within-subscription:
      +--ro module-version* [module-name]
         +--ro module-name       yang:yang-identifier
         +--ro revision          rev:revision-date-or-label
         +--ro revision-label?   ysver:version
    augment /sn:subscription-modified/sn:target/yp:datastore
              /yp:selection-filter/yp:within-subscription:
```

```
       +--ro module-version* [module-name]
          +--ro module-name       yang:yang-identifier
          +--ro revision          rev:revision-date-or-label
          +--ro revision-label?   ysver:version
     augment /sn:subscriptions/sn:subscription/sn:target/sn:stream
               /sn:stream-filter/sn:within-subscription:
       +--rw module-version* [module-name]
          +--rw module-name       yang:yang-identifier
          +--rw revision?         rev:revision-date-or-label
          +--rw revision-label?   ysver:version
     augment /sn:subscriptions/sn:subscription/sn:target/yp:datastore
               /yp:selection-filter/yp:within-subscription:
       +--rw module-version* [module-name]
          +--rw module-name       yang:yang-identifier
          +--rw revision?         rev:revision-date-or-label
          +--rw revision-label?   ysver:version
```

4.1.2. Full Tree View

The following is the YANG tree diagram [RFC8340] for the ietf-yang-push-revision augmentation within the ietf-subscribed-notifications, including the RPCs and notifications.

```
pyang -f tree ietf-yang-push-revision.yang ietf-yang-push@2019-09-09.yang ietf-ya
ng-revisions@2022-11-29.yang ietf-subscribed-notifications@2019-09-09.yang
module: ietf-yang-push

  rpcs:
    +---x resync-subscription {on-change}?
       +---w input
          +---w id    sn:subscription-id

  notifications:
    +---n push-update
    |  +--ro id?                    sn:subscription-id
    |  +--ro datastore-contents?    <anydata>
    |  +--ro incomplete-update?     empty
    +---n push-change-update {on-change}?
       +--ro id?                   sn:subscription-id
       +--ro datastore-changes
       |  +--ro yang-patch
       |     +--ro patch-id    string
       |     +--ro comment?    string
       |     +--ro edit* [edit-id]
       |        +--ro edit-id      string
       |        +--ro operation    enumeration
       |        +--ro target       target-resource-offset
       |        +--ro point?       target-resource-offset
       |        +--ro where?       enumeration
```

```
       |        +--ro value?       <anydata>
       +--ro incomplete-update?   empty

module: ietf-subscribed-notifications
  +--ro streams
  |  +--ro stream* [name]
  |     +--ro name                          string
  |     +--ro description?                  string
  |     +--ro replay-support?               empty {replay}?
  |     +--ro replay-log-creation-time      yang:date-and-time {replay}?
  |     +--ro replay-log-aged-time?         yang:date-and-time {replay}?
  +--rw filters
  |  +--rw stream-filter* [name]
  |  |  +--rw name                             string
  |  |  +--rw (filter-spec)?
  |  |  |  +--:(stream-subtree-filter)
  |  |  |  |  +--rw stream-subtree-filter?   <anydata> {subtree}?
  |  |  |  +--:(stream-xpath-filter)
  |  |  |     +--rw stream-xpath-filter?     yang:xpath1.0 {xpath}?
  |  |  +--rw ypr:module-version* [module-name]
  |  |     +--rw ypr:module-name       yang:yang-identifier
  |  |     +--rw ypr:revision?         rev:revision-date-or-label
  |  |     +--rw ypr:revision-label?   ysver:version
  |  +--rw yp:selection-filter* [filter-id]
  |     +--rw yp:filter-id                          string
  |     +--rw (yp:filter-spec)?
  |     |  +--:(yp:datastore-subtree-filter)
  |     |  |  +--rw yp:datastore-subtree-filter?   <anydata>
  |     |  |           {sn:subtree}?
  |     |  +--:(yp:datastore-xpath-filter)
  |     |     +--rw yp:datastore-xpath-filter?     yang:xpath1.0
  |     |              {sn:xpath}?
  |     +--rw ypr:module-version* [module-name]
  |        +--rw ypr:module-name       yang:yang-identifier
  |        +--rw ypr:revision?         rev:revision-date-or-label
  |        +--rw ypr:revision-label?   ysver:version
  +--rw subscriptions
     +--rw subscription* [id]
        +--rw id
        |       subscription-id
        +--rw (target)
        |  +--:(stream)
        |  |  +--rw (stream-filter)?
        |  |  |  +--:(by-reference)
        |  |  |  |  +--rw stream-filter-name
        |  |  |  |           stream-filter-ref
        |  |  |  +--:(within-subscription)
        |  |  |     +--rw (filter-spec)?
```

```
|  |  |  |  +--:(stream-subtree-filter)
|  |  |  |  |  +--rw stream-subtree-filter?
|  |  |  |  |          <anydata> {subtree}?
|  |  |  |  +--:(stream-xpath-filter)
|  |  |  |     +--rw stream-xpath-filter?
|  |  |  |             yang:xpath1.0 {xpath}?
|  |  |  +--rw ypr:module-version* [module-name]
|  |  |     +--rw ypr:module-name
|  |  |     |       yang:yang-identifier
|  |  |     +--rw ypr:revision?
|  |  |     |       rev:revision-date-or-label
|  |  |     +--rw ypr:revision-label?   ysver:version
|  |  +--rw stream
|  |  |       stream-ref
|  |  +--ro replay-start-time?
|  |  |       yang:date-and-time {replay}?
|  |  +--rw configured-replay?                          empty
|  |             {configured,replay}?
|  +--:(yp:datastore)
|     +--rw yp:datastore
|     |       identityref
|     +--rw (yp:selection-filter)?
|        +--:(yp:by-reference)
|        |  +--rw yp:selection-filter-ref
|        |          selection-filter-ref
|        +--:(yp:within-subscription)
|           +--rw (yp:filter-spec)?
|           |  +--:(yp:datastore-subtree-filter)
|           |  |  +--rw yp:datastore-subtree-filter?
|           |  |          <anydata> {sn:subtree}?
|           |  +--:(yp:datastore-xpath-filter)
|           |     +--rw yp:datastore-xpath-filter?
|           |             yang:xpath1.0 {sn:xpath}?
|           +--rw ypr:module-version* [module-name]
|              +--rw ypr:module-name
|              |       yang:yang-identifier
|              +--rw ypr:revision?
|              |       rev:revision-date-or-label
|              +--rw ypr:revision-label?   ysver:version
+--rw stop-time?
|       yang:date-and-time
+--rw dscp?
|       inet:dscp {dscp}?
+--rw weighting?                                       uint8
|       {qos}?
+--rw dependency?
|       subscription-id {qos}?
+--rw transport?
```

```
        |       transport {configured}?
        +--rw encoding?
        |       encoding
        +--rw purpose?                                          string
        |       {configured}?
        +--rw (notification-message-origin)? {configured}?
        |  +--:(interface-originated)
        |  |  +--rw source-interface?
        |  |          if:interface-ref {interface-designation}?
        |  +--:(address-originated)
        |     +--rw source-vrf?
        |     |       -> /ni:network-instances/network-instance/name
        |     |       {supports-vrf}?
        |     +--rw source-address?
        |             inet:ip-address-no-zone
        +--ro configured-subscription-state?
        |       enumeration {configured}?
        +--rw receivers
        |  +--rw receiver* [name]
        |     +--rw name                        string
        |     +--ro sent-event-records?
        |     |       yang:zero-based-counter64
        |     +--ro excluded-event-records?
        |     |       yang:zero-based-counter64
        |     +--ro state                       enumeration
        |     +---x reset {configured}?
        |        +--ro output
        |           +--ro time    yang:date-and-time
        +--rw (yp:update-trigger)?
           +--:(yp:periodic)
           |  +--rw yp:periodic!
           |     +--rw yp:period         centiseconds
           |     +--rw yp:anchor-time?   yang:date-and-time
           +--:(yp:on-change) {on-change}?
              +--rw yp:on-change!
                 +--rw yp:dampening-period?   centiseconds
                 +--rw yp:sync-on-start?      boolean
                 +--rw yp:excluded-change*    change-type

  rpcs:
    +---x establish-subscription
    |  +---w input
    |  |  +---w (target)
    |  |  |  +--:(stream)
    |  |  |  |  +---w (stream-filter)?
    |  |  |  |  |  +--:(by-reference)
    |  |  |  |  |  |  +---w stream-filter-name
    |  |  |  |  |  |          stream-filter-ref
```

```
      |  |  |  |  |  +--:(within-subscription)
      |  |  |  |  |     +---w (filter-spec)?
      |  |  |  |  |     |  +--:(stream-subtree-filter)
      |  |  |  |  |     |  |  +---w stream-subtree-filter?
      |  |  |  |  |     |  |           <anydata> {subtree}?
      |  |  |  |  |     |  +--:(stream-xpath-filter)
      |  |  |  |  |     |     +---w stream-xpath-filter?
      |  |  |  |  |     |              yang:xpath1.0 {xpath}?
      |  |  |  |  |     +---w ypr:module-version* [module-name]
      |  |  |  |  |        +---w ypr:module-name
      |  |  |  |  |        |       yang:yang-identifier
      |  |  |  |  |        +---w ypr:revision?
      |  |  |  |  |        |       rev:revision-date-or-label
      |  |  |  |  |        +---w ypr:revision-label?   ysver:version
      |  |  |  |  +---w stream
      |  |  |  |  |       stream-ref
      |  |  |  |  +---w replay-start-time?
      |  |  |  |          yang:date-and-time {replay}?
      |  |  |  +--:(yp:datastore)
      |  |  |     +---w yp:datastore
      |  |  |     |       identityref
      |  |  |     +---w (yp:selection-filter)?
      |  |  |        +--:(yp:by-reference)
      |  |  |        |  +---w yp:selection-filter-ref
      |  |  |        |          selection-filter-ref
      |  |  |        +--:(yp:within-subscription)
      |  |  |           +---w (yp:filter-spec)?
      |  |  |           |  +--:(yp:datastore-subtree-filter)
      |  |  |           |  |  +---w yp:datastore-subtree-filter?
      |  |  |           |  |           <anydata> {sn:subtree}?
      |  |  |           |  +--:(yp:datastore-xpath-filter)
      |  |  |           |     +---w yp:datastore-xpath-filter?
      |  |  |           |              yang:xpath1.0 {sn:xpath}?
      |  |  |           +---w ypr:module-version* [module-name]
      |  |  |              +---w ypr:module-name
      |  |  |              |       yang:yang-identifier
      |  |  |              +---w ypr:revision?
      |  |  |              |       rev:revision-date-or-label
      |  |  |              +---w ypr:revision-label?   ysver:version
      |  |  +---w stop-time?
      |  |  |       yang:date-and-time
      |  |  +---w dscp?
      |  |  |       inet:dscp {dscp}?
      |  |  +---w weighting?                                    uint8
      |  |  |       {qos}?
      |  |  +---w dependency?
      |  |  |       subscription-id {qos}?
      |  |  +---w encoding?
```

```
     |  |  |       encoding
     |  |  +---w (yp:update-trigger)?
     |  |     +--:(yp:periodic)
     |  |     |  +---w yp:periodic!
     |  |     |     +---w yp:period         centiseconds
     |  |     |     +---w yp:anchor-time?   yang:date-and-time
     |  |     +--:(yp:on-change) {on-change}?
     |  |        +---w yp:on-change!
     |  |           +---w yp:dampening-period?   centiseconds
     |  |           +---w yp:sync-on-start?      boolean
     |  |           +---w yp:excluded-change*    change-type
     |  +--ro output
     |     +--ro id                              subscription-id
     |     +--ro replay-start-time-revision?   yang:date-and-time
     |              {replay}?
     +---x modify-subscription
     |  +---w input
     |     +---w id
     |     |       subscription-id
     |     +---w (target)
     |     |  +--:(stream)
     |     |  |  +---w (stream-filter)?
     |     |  |     +--:(by-reference)
     |     |  |     |  +---w stream-filter-name
     |     |  |     |          stream-filter-ref
     |     |  |     +--:(within-subscription)
     |     |  |        +---w (filter-spec)?
     |     |  |        |  +--:(stream-subtree-filter)
     |     |  |        |  |  +---w stream-subtree-filter?
     |     |  |        |  |          <anydata> {subtree}?
     |     |  |        |  +--:(stream-xpath-filter)
     |     |  |        |     +---w stream-xpath-filter?
     |     |  |        |             yang:xpath1.0 {xpath}?
     |     |  |        +---w ypr:module-version* [module-name]
     |     |  |           +---w ypr:module-name
     |     |  |           |       yang:yang-identifier
     |     |  |           +---w ypr:revision?
     |     |  |           |       rev:revision-date-or-label
     |     |  |           +---w ypr:revision-label?   ysver:version
     |     |  +--:(yp:datastore)
     |     |     +---w yp:datastore
     |     |     |       identityref
     |     |     +---w (yp:selection-filter)?
     |     |        +--:(yp:by-reference)
     |     |        |  +---w yp:selection-filter-ref
     |     |        |          selection-filter-ref
     |     |        +--:(yp:within-subscription)
     |     |           +---w (yp:filter-spec)?
```

```
     |      |           |  +--:(yp:datastore-subtree-filter)
     |      |           |  |  +---w yp:datastore-subtree-filter?
     |      |           |  |           <anydata> {sn:subtree}?
     |      |           |  +--:(yp:datastore-xpath-filter)
     |      |           |     +---w yp:datastore-xpath-filter?
     |      |           |              yang:xpath1.0 {sn:xpath}?
     |      |         +---w ypr:module-version* [module-name]
     |      |            +---w ypr:module-name
     |      |            |       yang:yang-identifier
     |      |            +---w ypr:revision?
     |      |            |       rev:revision-date-or-label
     |      |            +---w ypr:revision-label?   ysver:version
     |      +---w stop-time?
     |      |       yang:date-and-time
     |      +---w (yp:update-trigger)?
     |         +--:(yp:periodic)
     |         |  +---w yp:periodic!
     |         |     +---w yp:period         centiseconds
     |         |     +---w yp:anchor-time?   yang:date-and-time
     |         +--:(yp:on-change) {on-change}?
     |            +---w yp:on-change!
     |               +---w yp:dampening-period?   centiseconds
     +---x delete-subscription
     |  +---w input
     |     +---w id    subscription-id
     +---x kill-subscription
        +---w input
           +---w id    subscription-id

  notifications:
    +---n replay-completed {replay}?
    |  +--ro id    subscription-id
    +---n subscription-completed {configured}?
    |  +--ro id    subscription-id
    +---n subscription-modified
    |  +--ro id
    |  |       subscription-id
    |  +--ro (target)
    |  |  +--:(stream)
    |  |  |  +--ro (stream-filter)?
    |  |  |  |  +--:(by-reference)
    |  |  |  |  |  +--ro stream-filter-name
    |  |  |  |  |           stream-filter-ref
    |  |  |  |  +--:(within-subscription)
    |  |  |  |     +--ro (filter-spec)?
    |  |  |  |     |  +--:(stream-subtree-filter)
    |  |  |  |     |  |  +--ro stream-subtree-filter?
    |  |  |  |     |  |           <anydata> {subtree}?
```

```
     |  |  |  |  |  +--:(stream-xpath-filter)
     |  |  |  |  |     +--ro stream-xpath-filter?
     |  |  |  |  |              yang:xpath1.0 {xpath}?
     |  |  |  |  +--ro ypr:module-version* [module-name]
     |  |  |  |     +--ro ypr:module-name       yang:yang-identifier
     |  |  |  |     +--ro ypr:revision
     |  |  |  |     |       rev:revision-date-or-label
     |  |  |  |     +--ro ypr:revision-label?   ysver:version
     |  |  |  +--ro stream
     |  |  |  |       stream-ref
     |  |  |  +--ro replay-start-time?
     |  |  |          yang:date-and-time {replay}?
     |  |  +--:(yp:datastore)
     |  |     +--ro yp:datastore
     |  |     |       identityref
     |  |     +--ro (yp:selection-filter)?
     |  |        +--:(yp:by-reference)
     |  |        |  +--ro yp:selection-filter-ref
     |  |        |          selection-filter-ref
     |  |        +--:(yp:within-subscription)
     |  |           +--ro (yp:filter-spec)?
     |  |           |  +--:(yp:datastore-subtree-filter)
     |  |           |  |  +--ro yp:datastore-subtree-filter?
     |  |           |  |          <anydata> {sn:subtree}?
     |  |           |  +--:(yp:datastore-xpath-filter)
     |  |           |     +--ro yp:datastore-xpath-filter?
     |  |           |              yang:xpath1.0 {sn:xpath}?
     |  |           +--ro ypr:module-version* [module-name]
     |  |              +--ro ypr:module-name       yang:yang-identifier
     |  |              +--ro ypr:revision
     |  |              |       rev:revision-date-or-label
     |  |              +--ro ypr:revision-label?   ysver:version
     |  +--ro stop-time?
     |  |       yang:date-and-time
     |  +--ro dscp?
     |  |       inet:dscp {dscp}?
     |  +--ro weighting?                                        uint8
     |  |       {qos}?
     |  +--ro dependency?
     |  |       subscription-id {qos}?
     |  +--ro transport?
     |  |       transport {configured}?
     |  +--ro encoding?                                         encoding
     |  +--ro purpose?                                          string
     |  |       {configured}?
     |  +--ro (yp:update-trigger)?
     |     +--:(yp:periodic)
     |     |  +--ro yp:periodic!
```

```
     |      |      +--ro yp:period         centiseconds
     |      |      +--ro yp:anchor-time?   yang:date-and-time
     |      +--:(yp:on-change) {on-change}?
     |         +--ro yp:on-change!
     |            +--ro yp:dampening-period?   centiseconds
     |            +--ro yp:sync-on-start?      boolean
     |            +--ro yp:excluded-change*    change-type
     +---n subscription-resumed
     |  +--ro id    subscription-id
     +---n subscription-started {configured}?
     |  +--ro id
     |  |       subscription-id
     |  +--ro (target)
     |  |  +--:(stream)
     |  |  |  +--ro (stream-filter)?
     |  |  |  |  +--:(by-reference)
     |  |  |  |  |  +--ro stream-filter-name
     |  |  |  |  |           stream-filter-ref
     |  |  |  |  +--:(within-subscription)
     |  |  |  |     +--ro (filter-spec)?
     |  |  |  |     |  +--:(stream-subtree-filter)
     |  |  |  |     |  |  +--ro stream-subtree-filter?
     |  |  |  |     |  |           <anydata> {subtree}?
     |  |  |  |     |  +--:(stream-xpath-filter)
     |  |  |  |     |     +--ro stream-xpath-filter?
     |  |  |  |     |              yang:xpath1.0 {xpath}?
     |  |  |  |     +--ro ypr:module-version* [module-name]
     |  |  |  |        +--ro ypr:module-name       yang:yang-identifier
     |  |  |  |        +--ro ypr:revision
     |  |  |  |        |       rev:revision-date-or-label
     |  |  |  |        +--ro ypr:revision-label?   ysver:version
     |  |  |  +--ro stream
     |  |  |  |       stream-ref
     |  |  |  +--ro replay-start-time?
     |  |  |  |       yang:date-and-time {replay}?
     |  |  |  +--ro replay-previous-event-time?
     |  |  |          yang:date-and-time {replay}?
     |  |  +--:(yp:datastore)
     |  |     +--ro yp:datastore
     |  |     |       identityref
     |  |     +--ro (yp:selection-filter)?
     |  |        +--:(yp:by-reference)
     |  |        |  +--ro yp:selection-filter-ref
     |  |        |           selection-filter-ref
     |  |        +--:(yp:within-subscription)
     |  |           +--ro (yp:filter-spec)?
     |  |           |  +--:(yp:datastore-subtree-filter)
     |  |           |  |  +--ro yp:datastore-subtree-filter?
```

```
     |  |           |  |          <anydata> {sn:subtree}?
     |  |           |  +--:(yp:datastore-xpath-filter)
     |  |           |     +--ro yp:datastore-xpath-filter?
     |  |           |              yang:xpath1.0 {sn:xpath}?
     |  |           +--ro ypr:module-version* [module-name]
     |  |              +--ro ypr:module-name       yang:yang-identifier
     |  |              +--ro ypr:revision
     |  |              |       rev:revision-date-or-label
     |  |              +--ro ypr:revision-label?   ysver:version
     |  +--ro stop-time?
     |  |       yang:date-and-time
     |  +--ro dscp?
     |  |       inet:dscp {dscp}?
     |  +--ro weighting?                                        uint8
     |  |       {qos}?
     |  +--ro dependency?
     |  |       subscription-id {qos}?
     |  +--ro transport?
     |  |       transport {configured}?
     |  +--ro encoding?                                         encoding
     |  +--ro purpose?                                          string
     |  |       {configured}?
     |  +--ro (yp:update-trigger)?
     |     +--:(yp:periodic)
     |     |  +--ro yp:periodic!
     |     |     +--ro yp:period         centiseconds
     |     |     +--ro yp:anchor-time?   yang:date-and-time
     |     +--:(yp:on-change) {on-change}?
     |        +--ro yp:on-change!
     |           +--ro yp:dampening-period?   centiseconds
     |           +--ro yp:sync-on-start?      boolean
     |           +--ro yp:excluded-change*    change-type
     +---n subscription-suspended
     |  +--ro id        subscription-id
     |  +--ro reason    identityref
     +---n subscription-terminated
        +--ro id        subscription-id
        +--ro reason    identityref
```

4.2. YANG Module

The YANG module has two leaves augmenting the model of Subscription to YANG Notifications [RFC8639].

```
<CODE BEGINS> file "ietf-yang-push-revision@2023-07-03.yang"
module ietf-yang-push-revision {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-yang-push-revision";
  prefix ypr;
  import ietf-subscribed-notifications {
    prefix sn;
    reference
      "RFC 8639: Subscription to YANG Notifications";
  }
  import ietf-yang-push {
    prefix yp;
    rev:recommended-min "2019-09-09";
    reference
      "RFC 8641: Subscriptions to YANG Datastores";
  }
  import ietf-yang-revisions {
    prefix rev;
    reference
      "RFC XXXX: draft-ietf-netmod-yang-module-versioning-06,
      Updated YANG Module Revision Handling";
  }
  import ietf-yang-types {
    prefix yang;
    rev:recommended-min "2013-07-15";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-yang-semver {
    prefix ysver;
    reference
      "RFC XXXX: draft-ietf-netmod-yang-semver-08, YANG Semantic
      Versioning";
  }

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

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

```
  description
    "Defines YANG push event notification header with the revision and
    the revision-label. Adds the support of the revision and
    revision-label selection in the YANG push subscription RPCs.

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

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

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

  revision 2023-07-03 {
    description
      "First revision";
    reference
      "RFC XXXX: Support of Versioning in YANG Notifications
      Subscription";
  }

  feature yang-push-revision-supported {
    description
      "This feature indicates the YANG Subscription Notifications
      supports specifying the list of modules, revisions and
      revision-label in the YANG subscription.";
  }

  grouping yang-push-module-version-config {
    description
      "This grouping combines the module name, the revision and
      revision-label leaves, with the revision being configurable.";
    leaf module-name {
      type yang:yang-identifier;
      description
        "This references the YANG module name.";
    }
    leaf revision {
      type rev:revision-date-or-label;
      description
        "This references the YANG module revision to be sent in the
        subscription.";
    }
    leaf revision-label {
```

```
        type ysver:version;
        description
          "This references the YANG module semversion to be sent in the
          subscription.";
      }
    }

    grouping yang-push-module-version {
      description
        "This grouping combines the module name, the revision and
        revision-label leaves, with the revision being non
        configurable.";
      leaf module-name {
        type yang:yang-identifier;
        config false;
        mandatory true;
        description
          "This references the YANG module name.";
      }
      leaf revision {
        type rev:revision-date-or-label;
        config false;
        mandatory true;
        description
          "This references the YANG module revision of the sent
          notification message.";
      }
      leaf revision-label {
        type ysver:version;
        description
          "This references the YANG module semversion of the sent
          notification message.";
      }
    }

    grouping yang-push-module-version-list {
      description
        "This grouping defines a list of yang-push-module-version
        grouping.";
      list module-version {
        key "module-name";
        config false;
        description
          "List of yang-push-module-version grouping. The revision is
          not configurable.";
        uses ypr:yang-push-module-version;
      }
    }
```

```
  grouping yang-push-module-version-config-list {
    description
      "This grouping defines a list of yang-push-module-version-config
      grouping.";
    list module-version {
      key "module-name";
      description
        "List of yang-push-module-version-config grouping. The revision
        is configurable.";
      uses ypr:yang-push-module-version-config;
    }
  }

  // Filters
  augment "/sn:filters/sn:stream-filter" {
    description
      "Augment the filters from subscribed notifications with a list
      of yang-push-module-version-config grouping.";
    uses ypr:yang-push-module-version-config-list;
  }
  augment "/sn:filters/yp:selection-filter" {
    description
      "Augment the filters from YANG push with a list of
      yang-push-module-version grouping.";
    uses ypr:yang-push-module-version-config-list;
  }

  // Subscription parameters
  augment "/sn:establish-subscription/sn:input/sn:target/sn:stream"
        + "/sn:stream-filter/sn:within-subscription" {
    description
      "Augment the establish-subscription RPC from the
      ietf-subscribed-notifications YANG module with the
      yang-push-module-version-config-list grouping.";
    uses ypr:yang-push-module-version-config-list;
  }
  augment "/sn:establish-subscription/sn:input/sn:target"
        + "/yp:datastore/yp:selection-filter/yp:within-subscription" {
    description
      "Augment the establish-subscription RPC from the ietf-yang-push
      YANG module with the yang-push-module-version-config-list
      grouping.";
    uses ypr:yang-push-module-version-config-list;
  }
  augment "/sn:modify-subscription/sn:input/sn:target/sn:stream"
        + "/sn:stream-filter/sn:within-subscription" {
    description
      "Augment the modify-subscription RPC from the
```

```
        ietf-subscribed-notifications YANG module with the
        yang-push-module-version-config-list grouping.";
      uses ypr:yang-push-module-version-config-list;
    }
    augment "/sn:modify-subscription/sn:input/sn:target/yp:datastore"
          + "/yp:selection-filter/yp:within-subscription" {
      description
        "Augment the modify-subscription RPC from the ietf-yang-push
        YANG module with the yang-push-module-version-config-list
        grouping.";
      uses ypr:yang-push-module-version-config-list;
    }

    // Subscription notifications
    augment "/sn:subscription-started/sn:target/sn:stream"
          + "/sn:stream-filter/sn:within-subscription" {
      description
        "Augment the subscription-started notification from the
        ietf-subscribed-notifications YANG module with the
        yang-push-module-version-list grouping.";
      uses ypr:yang-push-module-version-list;
    }
    augment "/sn:subscription-started/sn:target/yp:datastore"
          + "/yp:selection-filter/yp:within-subscription" {
      description
        "Augment the subscription-started notification from the
        ietf-yang-push YANG module with the
        yang-push-module-version-list grouping.";
      uses ypr:yang-push-module-version-list;
    }
    augment "/sn:subscription-modified/sn:target/sn:stream"
          + "/sn:stream-filter/sn:within-subscription" {
      description
        "Augment the subscription-modified notification from the
        ietf-subscribed-notifications YANG module with the
        yang-push-module-version-list grouping.";
      uses ypr:yang-push-module-version-list;
    }
    augment "/sn:subscription-modified/sn:target/yp:datastore"
          + "/yp:selection-filter/yp:within-subscription" {
      description
        "Augment the subscription-modified notification from the
        ietf-yang-push YANG module with the
        yang-push-module-version-list grouping.";
      uses ypr:yang-push-module-version-list;
    }

    // Subscription container
```

```
    augment "/sn:subscriptions/sn:subscription/sn:target/sn:stream"
          + "/sn:stream-filter/sn:within-subscription" {
      description
        "Augment the subscriptions RPC container from the
        ietf-subscribed-notifications YANG module with the
        yang-push-module-version-config-list grouping.";
      uses ypr:yang-push-module-version-config-list;
    }
    augment "/sn:subscriptions/sn:subscription/sn:target/yp:datastore"
          + "/yp:selection-filter/yp:within-subscription" {
      description
        "Augment the subscription RPC container from ietf-yang-push
        YANG module with the yang-push-module-version-config-list grouping.";
      uses ypr:yang-push-module-version-config-list;
    }
  }
  <CODE ENDS>
```

5. Security Considerations

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

6. IANA Considerations

This document has no IANA actions.

7. Acknowledgements

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

8. References

8.1. Normative References

[I-D.ietf-netmod-yang-module-versioning]
Wilton, R., Rahman, R., Lengyel, B., Clarke, J., and J. Sterne, "Updated YANG Module Revision Handling", Work in Progress, Internet-Draft, draft-ietf-netmod-yang-module-versioning-09, 17 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-module-versioning-09>.

[I-D.ietf-netmod-yang-semver]
Clarke, J., Wilton, R., Rahman, R., Lengyel, B., Sterne, J., and B. Claise, "YANG Semantic Versioning", Work in

Progress, Internet-Draft, draft-ietf-netmod-yang-semver-11, 10 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-yang-semver-11>.

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

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

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

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

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

8.2. Informative References

[I-D.ietf-netconf-https-notif]
Jethanandani, M. and K. Watsen, "An HTTPS-based Transport for YANG Notifications", Work in Progress, Internet-Draft, draft-ietf-netconf-https-notif-13, 4 November 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-https-notif-13>.

[I-D.ietf-netconf-udp-notif]
Zheng, G., Zhou, T., Graf, T., Francois, P., Feng, A. H., and P. Lucente, "UDP-based Transport for Configured Subscriptions", Work in Progress, Internet-Draft, draft-ietf-netconf-udp-notif-09, 10 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-udp-notif-09>.

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

[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>.

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

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

Authors' Addresses

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

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 11 January 2024

JG. Cumming
Nokia
R. Wills
Cisco Systems

10 July 2023

NETCONF Private Candidates
draft-jgc-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 11 January 2024.

Copyright Notice

Table of Contents

1. Introduction

NETCONF [RFC6241] and RESTCONF [RFC8040] both provide a mechanism for one or more clients to make configuration changes to a device running as a NETCONF/RESTCONF server. Each client has the ability to make one or more configuration change to the servers shared candidate configuration.

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

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

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

This document details the extensions to the NETCONF protocol in order to support the use of private candidates. It also describes how the RESTCONF protocol can be used on a system that implements private candidates.

1.1. Requirements Language

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

2. Definitions and terminology

2.1. Session specific datastore

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

2.2. Shared candidate configuration

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

2.3. Private candidate configuration

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

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

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

The private candidate configuration contains a full copy of the running configuration when it is created (in the same way as a branch does in a source control management system and in the same way as the candidate configuration datastore as defined in [RFC6241]). Any changes made to it, for example, through the use of operations such as <edit-config> and <edit-data>, are made in this private candidate configuration.

Obtaining this private candidate over NETCONF will display the entire configuration, including all changes made to it. Performing a <commit> operation will merge the changes from the private candidate into the running configuration (the same as a merge in source code management systems). A <discard-changes> operation will revert the private candidate to the branch's initial state.

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

3. Limitations using the shared candidate configuration for multiple clients

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

3.1. Issues

3.1.1. Unintended deployment of alternate users configuration changes

Consider the following scenario:

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

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

3. Client 2 then issues a <commit> RPC

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

3.2. Current mitigation strategies

3.2.1. Locking the shared candidate configuration datastore

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

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

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

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

3.2.2. Always use the running configuration datastore

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

3.2.3. Fine-grained locking

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

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

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

4. Private candidates solution

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

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

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

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

* Be accessible by one client only

* Be visible by one client only

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

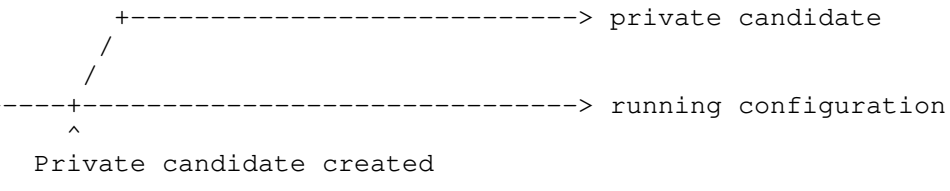
4.1. What is a private candidate

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

4.2. When is a private candidate created

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

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



Closing a session that is operating using a private candidate will discard all changes in that session's private candidate and destroy the private candidate.

4.3. How to signal the use of private candidates

4.3.1. Server

The server MUST signal its support for private candidates. The server does this by advertising the :candidate capability as defined in [RFC6241] and a new :private-candidate capability:

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

If the server has not signalled these capabilities, or has signalled the support for :candidate but not :private-candidate, or :private-candidate but not :candidate the session MUST be terminated when a client attempts to interact with a private candidate (for example, by explicitly targeting the private-candidate NMDA datastore).

4.3.2. 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:

4.3.2.1. 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 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>)

4.3.2.2. 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.

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

4.3.3. RESTCONF client

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

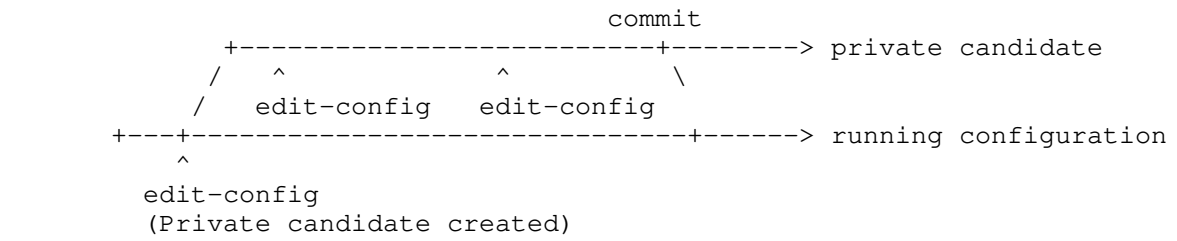
When the server advertises the :private-candidate capability and the client does not explicitly reference a datastore in their request, all edits are made to a new private candidate, and the private candidate is committed. This is analagous to the behavior of RESTCONF when the :candidate capability is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

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

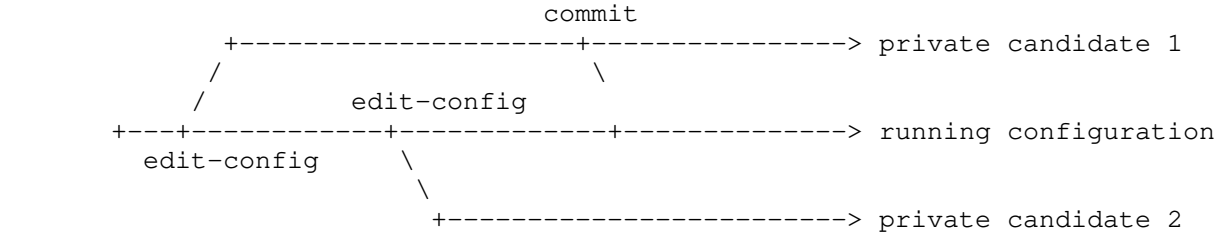
4.4. Interaction between running and private-candidate(s)

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

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



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



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

As described earlier, the client MUST be aware of changes to the running configuration so it can be assured that it is only committing its own modifications.

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

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

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

4.4.1. Static branch mode: Independent private candidate branch

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

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

* <commit> is issued (which SHOULD automatically issue an <update> operation)

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

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

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

Conflict detection and resolution is discussed later in this document.

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

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

4.4.2. Continuous rebase mode: Continually updating private candidate

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

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

In this model of operation the following should be considered:

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

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

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

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

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

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

Conflict detection and resolution is discussed later in this document.

4.5. Detecting and resolving conflicts

4.5.1. What is a conflict?

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

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

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

Examples of conflicts include:

* An interface has been deleted in the running configuration that existed when the private candidate was created. A change to a child node of this specific interface is made in the private

candidate using the default merge operation would, instead of changing the child node, both recreate the interface and then set the child node.

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

4.5.2. Detecting and reporting conflicts

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

* Manually triggered by the <update> NETCONF operation

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

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

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

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

In continuous rebase mode, it is possible for the automated <update> operation to fail. In this instance, the next NETCONF operation (of any type) MUST fail. It MUST inform the client of the conflict and SHOULD detail the location of the conflict(s).

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

4.5.3. 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, the client MUST be informed. The client then has a number of options available to resolve the conflict.

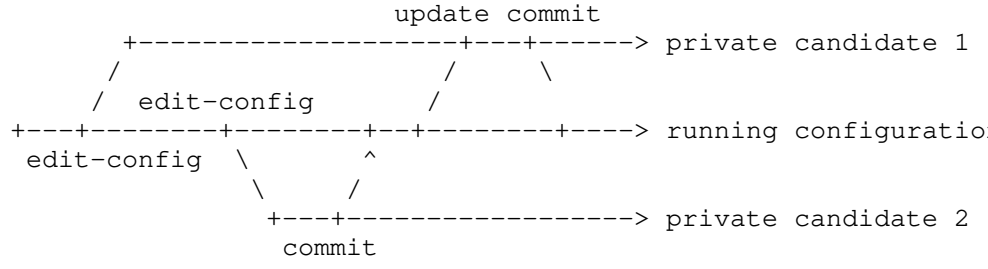
It is worth noting that in the case of continuous rebase mode automated <update> operations may be performed against multiple private candidate configurations at once.

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

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

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

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



There are three defined resolution methods:

4.5.3.1. 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 committing 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>
```

4.5.3.2. 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 committing 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>
```

4.5.3.3. Revert-on-conflict

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

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

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

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

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

Example:

Session 1 edits the configuration by submitting the following

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

Session 2 then edits the configuration deleting the interface intf_one, updating the description on interface intf_two and committing 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 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>
```

4.5.4. Default resolution mode and advertisement of this mode

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

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

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

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

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

4.5.5. Supported resolution modes

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

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

4.6. NETCONF operations

4.6.1. New NETCONF operations

4.6.1.1. <update>

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

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

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

The <update> operation SHOULD be triggered by a specific NETCONF session issuing a <commit> operation.

4.6.1.1.1. <resolution-mode> parameter

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

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

* revert-on-conflict (default)
* ignore
* overwrite

4.6.2. 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.

4.6.2.1. <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.

4.6.2.2. <lock> and <unlock>

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

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

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

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

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

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

4.6.2.3. <compare>

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

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

The <compare> operation 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. This document adds the optional <reference-point> node to the input of the <compare> operation that accepts the values of last-update or 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.

4.6.2.4. <get-config>

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

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

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

4.6.2.5. <get-data>

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

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

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

4.6.2.6. <copy-config>

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

4.6.2.7. <delete-config>

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

4.6.2.8. <commit>

The <commit> operation SHOULD trigger an <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 memo includes no request to IANA.

6. Security Considerations

This document should not affect the security of the Internet.

7. References

7.1. Normative References

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

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

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

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

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

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

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

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

7.2. Informative References

Appendix A. Behavior with unaltered NETCONF operations

A.1. <get>

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

Contributors

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

Authors' Addresses

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

Robert Wills
Cisco Systems
Email: rowills@cisco.com

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: 11 January 2024

J. Quilbeuf
B. Claise
Huawei
T. Graf
Swisscom
D. Lopez
Telefonica I+D
Q. Sun
China Telecom
10 July 2023

External Trace ID for Configuration Tracing
draft-quilbeuf-opsawg-configuration-tracing-02

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 [I-D.ietf-opsawg-service-assurance-architecture], is able to detect and report network anomalies , e.g. SLA violations, intent violations, network failure, or simply a customer issue. Here are the use cases for the proposed YANG module.

3.1. Configuration Mistakes

Taking into account that many network anomalies are due to configuration mistakes, this mechanism allows to find out whether the offending configuration modification was triggered by a tracing-enabled client/NMS. In such 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.

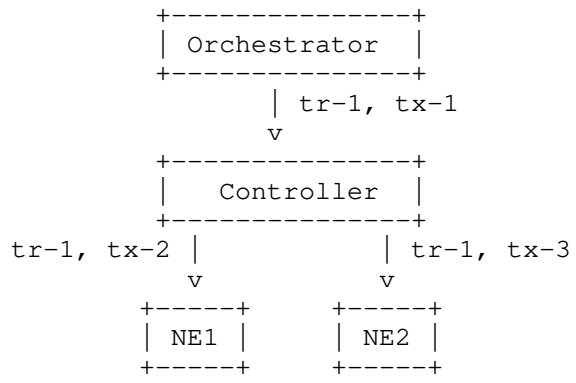


Figure 1: Example of Hierarchical Configuration. tx: transaction. tr: trace.

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.

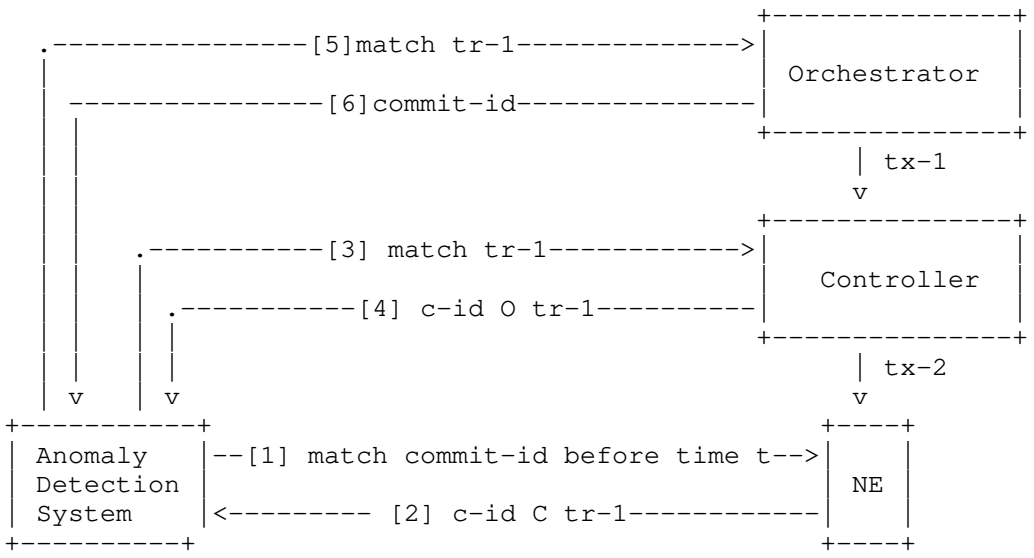


Figure 2: Example of Configuration Tracing. tr: trace-id, C: Controller, O: orchestrator. The number between square brackets refer to steps in the listing below.

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

[I-D.ietf-opsawg-service-assurance-architecture]
Claise, B., Quilbeuf, J., Lopez, D., Voyer, D., and T. Arumugam, "Service Assurance for Intent-based Networking Architecture", Work in Progress, Internet-Draft, draft-ietf-opsawg-service-assurance-architecture-13, 3 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-service-assurance-architecture-13>.

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

Appendix A. Changes between revisions

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

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: 7 January 2024

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

6 July 2023

RESTCONF Extension to support Trace Context Headers
draft-rogaglia-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-rogaglia-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 7 January 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/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>.

[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>.

[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/info/rfc8309>.

Appendix A. Example RESTCONF calls

TBD

Appendix B. Changes (to be deleted by RFC Editor)

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: 7 January 2024

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

6 July 2023

NETCONF Extension to support Trace Context propagation
draft-rogaglia-netconf-trace-ctx-extension-03

Abstract

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

About This Document

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

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

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

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

Status of This Memo

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

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

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.

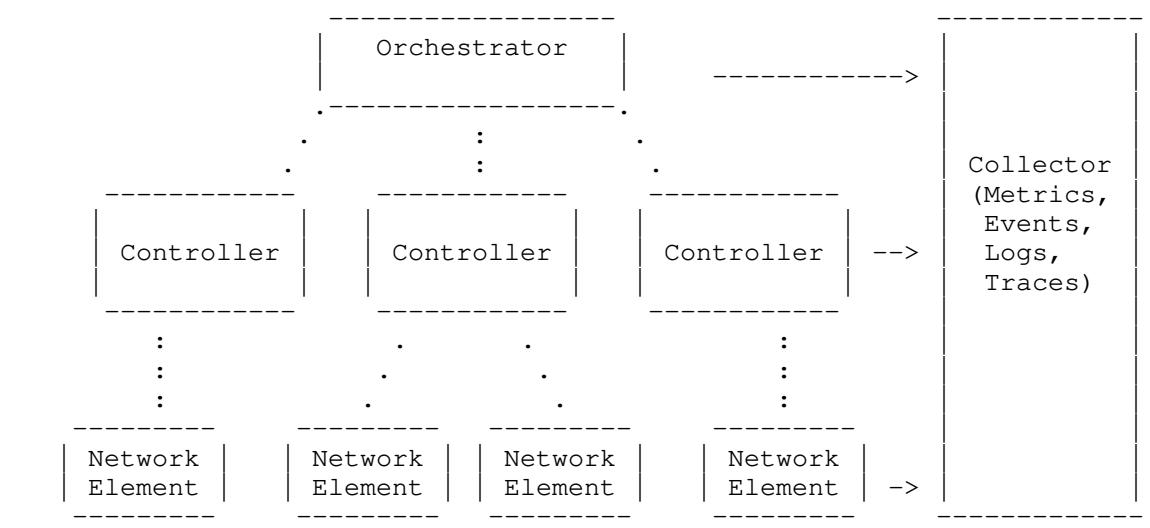


Figure 1: A Sample SDN Architecture from RFC8309 augmented to include the export of metrics, events, logs and traces from the different components to a common collector.

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.

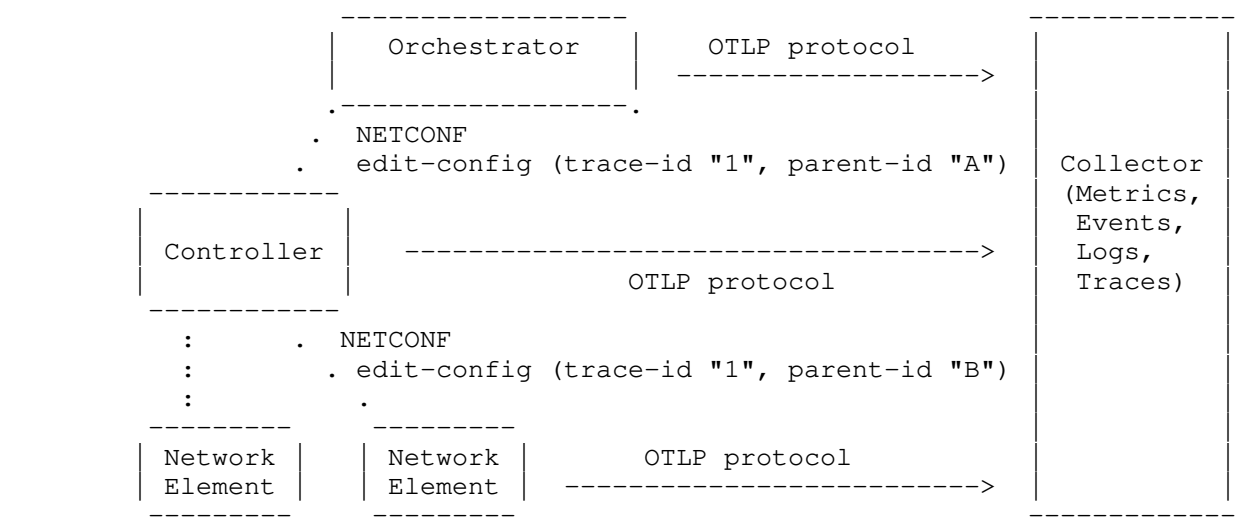


Figure 2: An implementation example where the NETCONF protocol is used between the Orchestrator and the Controller and also between the Controller and the Network Elements. Every component exports M.E.L.T information to the collector using the OTLP protocol.

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.

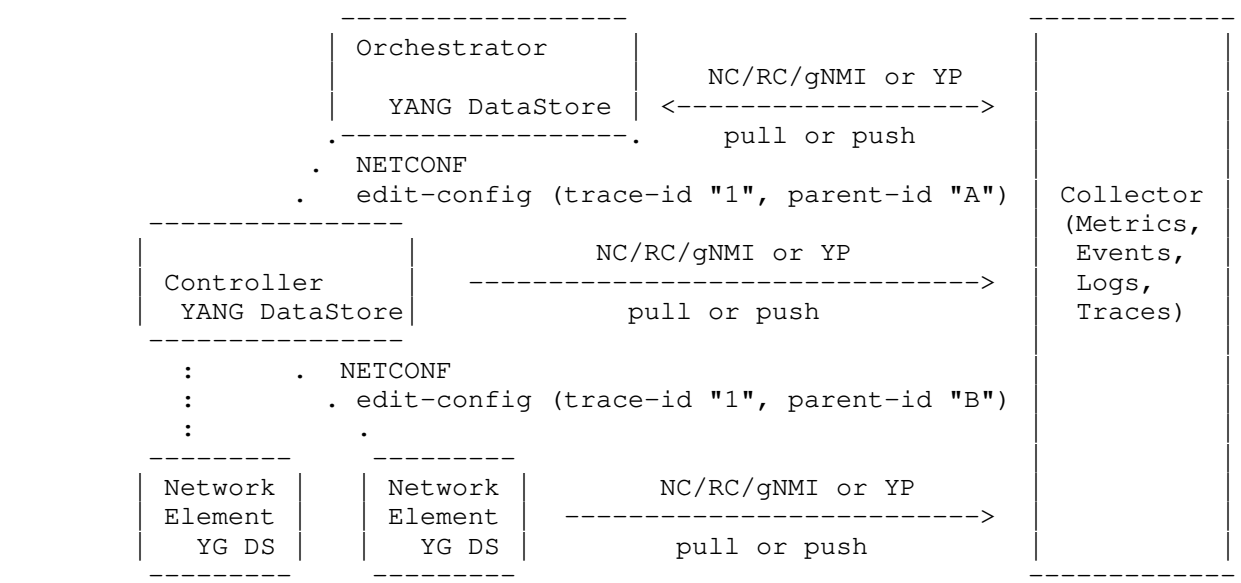


Figure 3: An implementation example where the NETCONF protocol is used between the Orchestrator and the Controller and also between the Controller and the Network Elements. M.E.L.T. information is stored in local Yang Datastores and accessed by the collector using "pull" mechanisms using the NETCONF (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/info/rfc2119>.

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

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

[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>.

[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-01, 4 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-netconf-transaction-id-01>.

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

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

[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>.

[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 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.2. 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.3. 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