NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

A. Gonzalez Prieto
VMware
A. Clemm
Huawei
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems

July 3, 2017

NETCONF Support for Event Notifications
draft-ietf-netconf-netconf-event-notifications-04

Abstract

This document defines how to transport network subscriptions and event messages on top of the Network Configuration protocol (NETCONF). This includes the full set of RPCs, subscription state changes, and message payloads needing asynchronous delivery. The capabilities and operations defined in this document used in conjunction with [subscribe] are intended to obsolete [RFC5277]. In addition, the capabilities within those two documents along with [yang-push] are intended to enable an extract of a YANG datastore on a remote device.

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 http://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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service for the NETCONF protocol [RFC6241] based on [subscribe]. This is an optional capability built on top of the base NETCONF definition.

The document [subscribe] plus this transport specification document provides a superset of the capabilities previously defined in [RFC5277]. Newly introduced capabilities include the ability to have multiple subscriptions on a single NETCONF session, the ability to terminate subscriptions without terminating the client session, and the ability to modify existing subscriptions.

In addition, [yang-push] plus the capabilities of this document provide a mechanism for a NETCONF client to maintain a subset/extract of an actively changing YANG datastore.

2. Terminology

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

The following terms are defined in [RFC6241]: client, server, operation, RPC.

The following terms are defined in [subscribe]: event, event notification, stream, publisher, receiver, subscriber, subscription, configured subscription.

Note that a publisher in [subscribe] corresponds to a server in [RFC6241]. Similarly, a subscriber corresponds to a client. A receiver is also a client. In the remainder of this document, we will use the terminology in [RFC6241] to simplify [subscribe]'s mental mappings to embedded NETCONF terminology.

3. Solution

In this section, we describe and exemplify how [subscribe] is to be supported over NETCONF.

3.1. Event Stream Discovery

In the context of [subscribe] an event stream exposes a continuous set of events available for subscription. A NETCONF client can retrieve the list of available event streams from a NETCONF server using the "get" operation against the top-level container "/streams" defined in [subscribe]. The reply includes the stream identities supported on the NETCONF server.

The following example illustrates the retrieval of the list of available event streams using the <get> operation.

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <streams
      xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0"/>
      </filter>
  </get>
</rpc>
```

Figure 1: Get streams request

The NETCONF server returns a list of event streams available. In this example, the list contains the NETCONF, SNMP, and SYSLOG streams.

```
<rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <streams
       xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <stream>NETCONF</stream>
      <stream>SNMP</stream>
      <stream>SYSLOG</stream>
      </streams>
  </data>
</rpc-reply>
```

Figure 2: Get streams response

For [yang-push], a similar get is needed to retrieve available datastore names.

3.2. Mandatory NETCONF support

A NETCONF server implementation supporting [subscribe] must support dynamic subscriptions and the "NETCONF" notification event stream. The NETCONF event stream contains all NETCONF XML event information supported by the server, except for where it has been explicitly indicated that this the event must be excluded from the NETCONF stream.

A NETCONF server implementation supporting [yang-push] must support the "running" datastore.

3.3. Dynamic Subscriptions

3.3.1. Establishing Dynamic Subscriptions

The dynamic subscription RFC and interactions operation is defined in [subscribe].

3.3.1.1. Usage Example

An example of interactions over NETCONF transport for one sample subscription is below:

```
  <netconf:rpc netconf:message-id="102"
         xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
    <establish-subscription
           xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <stream>NETCONF</stream>
      <event-filter-type>xpath</event-filter-type>
      <event-filter> xmlns:ex="http://example.com/event/1.0"
             select="/ex:event[ex:eventClass='fault' and
                      (ex:severity='minor' or ex:severity='major'
                       or ex:severity='critical')]"
      </event-filter>
    </establish-subscription>
  </netconf:rpc>
```

Figure 3: establish-subscription over NETCONF

3.3.1.2. Positive Response

If the NETCONF server can satisfy the request, the server sends a positive <subscription-result> element, and the subscription-id of the accepted subscription.

```
   <rpc-reply message-id="102"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <subscription-result
           xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
       ok
     </subscription-result>
     <identifier
           xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
       52
     </identifier>
   </rpc-reply>
```

Figure 4: Successful establish-subscription

3.3.1.3. Negative Response

If the NETCONF server cannot satisfy the request, or client has no authorization to establish the subscription, the server will send a negative <subscription-result> element. For instance:

```
<rpc-reply message-id="103"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
          xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    stream-unavailable
  </subscription-result>
</rpc-reply>
```

Figure 5: Unsuccessful establish subscription

3.3.1.4. Subscription Negotiation

If the client requests parameters the NETCONF server cannot serve, the negative <subscription-result> may include hints at subscription parameters which would have been accepted. For instance, consider the following subscription from [yang-push], which augments the establish-subscription with some additional parameters, including "period". If the client requests a period which the NETCONF server cannot serve, the back-and-forth exchange may be:

```
    <netconf:rpc message-id="101"
             xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
      <establish-subscription
               xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
        <stream>push-update</stream>
     <event-filter-type>subtree</event-filter-type>
           <event-filter>xmlns:ex="http://example.com/sample-data/1.0"
                       select="/ex:foo"</event-filter>
        <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
          500
        </period>
        <encoding>encode-xml</encoding>
      </establish-subscription>
    </netconf:rpc>

    <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <subscription-result
            xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
        error-insufficient-resources
      </subscription-result>
      <period
            xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
        2000
      </period>
    </rpc-reply>
```

Figure 6: Subscription establishment negotiation

3.3.1.5. Message Flow Examples

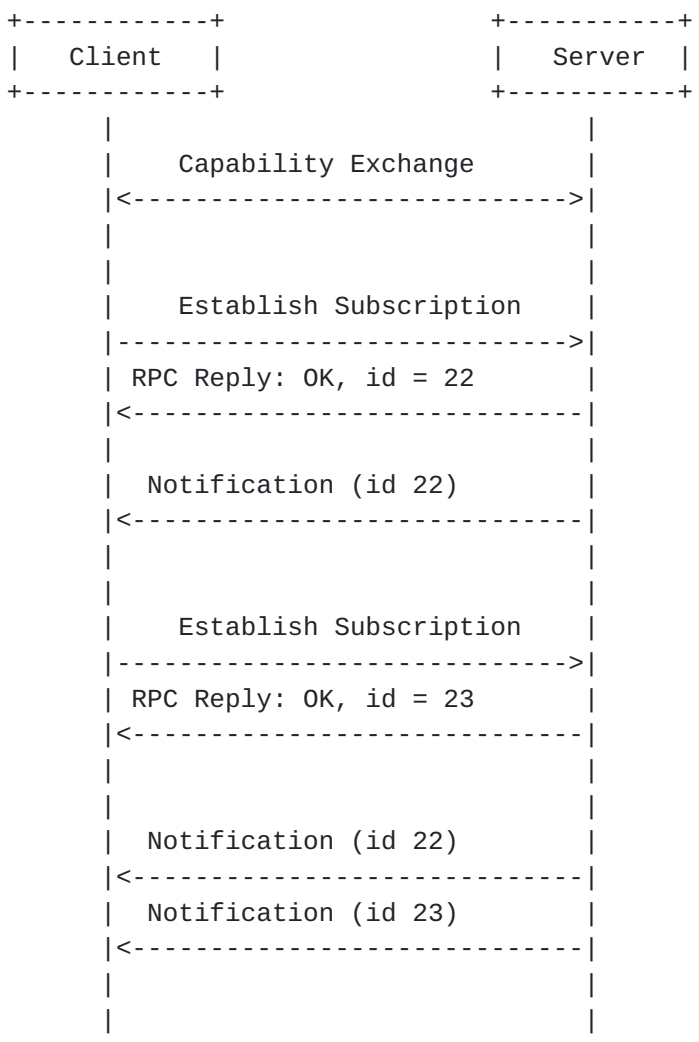

Figure 7: Multiple subscription establishments over a single NETCONF session

3.3.2. Modifying a Subscription

This operation is defined in [subscribe] and enhanced in [yang-push].

3.3.2.1. Usage Example

The following demonstrates modifying a subscription. Consider a subscription from [yang-push], which augments the establish-subscription with some additional parameters, including "period". A subscription may be established and modified as follows.

```
   <netconf:rpc message-id="101"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
     <establish-subscription
             xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
        <datastore xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">running</
datastore>
        <event-filter-type>subtree</event-filter-type>
        <event-filter>xmlns:ex="http://example.com/sample-data/1.0"
               select="/ex:foo"</event-filter>
        <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
            1000
        </period>
     </establish-subscription>
   </netconf:rpc>

   <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <subscription-result
             xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
       ok
     </subscription-result>
     <identifier
             xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
       1922
     </identifier>
   </rpc-reply>


   <netconf:rpc message-id="102"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
     <modify-subscription
             xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
       <identifier>1922</identifier>
       <period>100</period>
     </modify-subscription>
   </netconf:rpc>
```

Figure 8: Subscription modification

3.3.2.2. Positive Response

If the NETCONF server can satisfy the request, the server sends a positive <subscription-result> element. This response is like that to an establish-subscription request, but without the subscription identifier.

```
<rpc-reply message-id="102"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
          xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    ok
  </subscription-result>
</rpc-reply>
```

Figure 9: Successful modify subscription

3.3.2.3. Negative Response

If the NETCONF server cannot satisfy the request, the server sends a negative <subscription-result> element. Its contents and semantics are identical to those in an establish-subscription request. For instance:

```
<rpc-reply message-id="102"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    period-unsupported
  </subscription-result>
  <period-hint>500</period-hint>
</rpc-reply>
```

Figure 10: Unsuccessful modify subscription

3.3.2.4. Message Flow Example

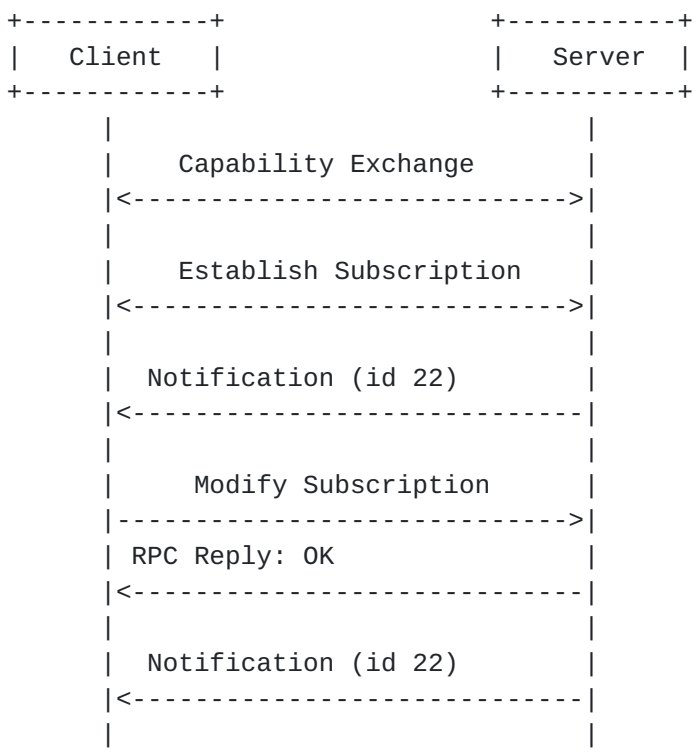

Figure 11: Message flow for successful subscription modification

3.3.3. Deleting a Subscription

This operation is defined in [subscribe] for events, and enhanced in [yang-push] for datastores.

3.3.3.1. Usage Example

The following demonstrates deleting a subscription.

```
<netconf:rpc message-id="101"
       xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
          xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <identifier>1922</identifier>
  </delete-subscription>
</netconf:rpc>
```

Figure 12: Delete subscription

3.3.3.2. Positive Response

If the NETCONF server can satisfy the request, the server sends an OK element. For example:

```
    <rpc-reply message-id="103"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <ok/>
    </rpc-reply>
```

Figure 13: Successful delete subscription

3.3.3.3. Negative Response

If the NETCONF server cannot satisfy the request, the server sends an error-rpc element indicating the modification didn't work. For example:

```
<rpc-reply message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path
           xmlns:t="urn:ietf:params:xml:ns:netconf:notification:2.0">
      /t:identifier
    </error-path>
    <error-message xml:lang="en">
      no-such-subscription
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 14: Unsuccessful delete subscription

3.4. Configured Subscriptions

Configured subscriptions are established, modified, and deleted using configuration operations against the top-level subtree of [subscribe] or [yang-push] via configuration interface. In this document, we focus on NETCONF operations. Any other configuration interface can be used to establish a configured subscription that uses NETCONF to push notifications to receivers. Configured susbscriptions are supported by NETCONF servers using NETCONF Call Home [RFC8071]. Note that this document only covers configured subscriptions where the protocol of choice is NETCONF. In this section, we present examples of how to manage the configuration subscriptions using a NETCONF client. Key differences from dynamic subscriptions over NETCONF is that subscription lifetimes are decoupled from NETCONF sessions.

3.4.1. Establishing a Configured Subscription

For subscription establishment, a NETCONF client may send:

```
 <rpc message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
        xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
   <edit-config>
     <target>
       <running/>
     </target>
     <subscription-config
           xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
       <subscription>
         <identifier>
           1922
         </identifier>
         <stream>
           foo
         </stream>
         <receiver>
           <address>
             1.2.3.4
           </address>
           <port>
             1234
           </port>
           <protocol>
             netconf
           </protocol>
         </receiver>
       </subscription>
     </subscription-config>
   </edit-config>
 </rpc>
```

Figure 15: Establish configured subscription

if the request is accepted, the server would reply:

```
    <rpc-reply message-id="101"
               xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <ok/>
    </rpc-reply>
```

Figure 16: Response to a successful configuration subscription establishment

if the request is not accepted because the server cannot serve it, no configuration is changed. Int this case the server may reply:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
      <error-tag>resource-denied</error-tag>
      <error-severity>error&lt;/error-severity>
      <error-message xml:lang="en">
        Temporarily the server cannot serve this
        subscription due to the current workload.
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 17: Response to a failed configured subscription establishment

For every configured receiver, once NETCONF transport session between the server and the receiver is recognized as active, the server will issue a "subscription-started" notification. After that, the server will send notifications to the receiver as per the subscription notification. The session is only intended for pushing notifications. Client request on that session SHOULD be ignored by the server.

The contents sent by the server on the Call Home session, once established, are identical to those in a dynamic subscription.

Note that the server assumes that the receiver is ready to accept notifications on the NETCONF session. This may require coordination between the client that configures the subscription and the clients for which the notifications are intended. This coordination is out of the scope of this document.

3.4.2. Call Home for Configured Subscriptions

Once this configuration is active, if NETCONF transport is needed but does not exist to one or more target IP address plus port, the server

initiates a transport session via [RFC8071] to those receiver(s) in the subscription using the address and port specified.

3.4.3. Full Establish Message Flow

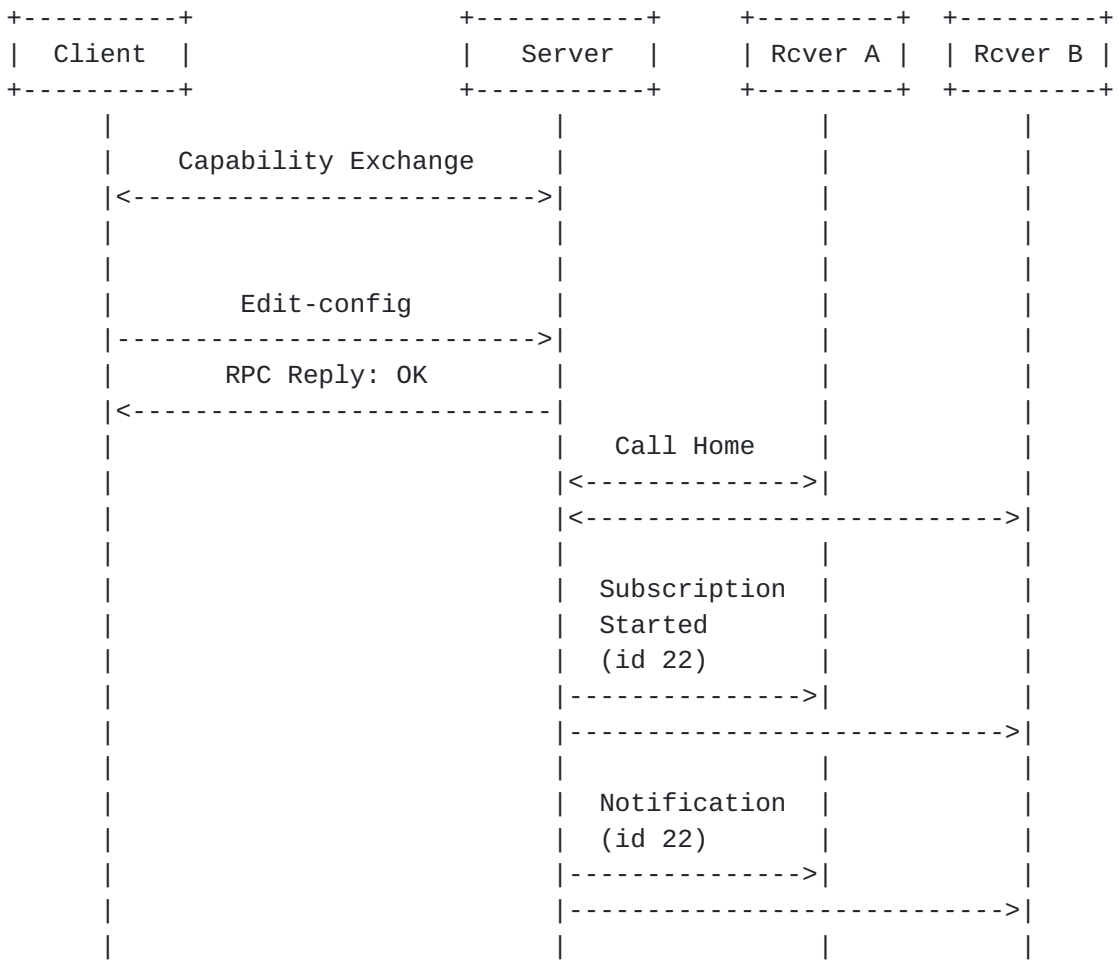

Figure 18: Message flow for configured subscription establishment

3.4.4. Modifying a Configured Subscription

Configured subscriptions can be modified using configuration operations against the top-level subtree subscription-config.

For example, the subscription established in the previous section could be modified as follows, choosing a different receiver:

```
    <rpc message-id="102"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <edit-config>
        <target>
          <running/>
        </target>
        <subscription-config
                xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
          <subscription>
            <identifier>
              1922
            </identifier>
            <stream>
              foo
            </stream>
            <receiver>
              <address>
                1.2.3.5
              </address>
              <port>
                1234
              </port>
            </receiver>
          </subscription>
        </subscription-config>
      </edit-config>
    </rpc>
```

Figure 19: Modify configured subscription

if the request is accepted, the server would reply:

```
    <rpc-reply message-id="102"
              xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <ok/>
    </rpc-reply>
```

Figure 20: A successful configured subscription modification

3.4.4.1. Message Flow Example

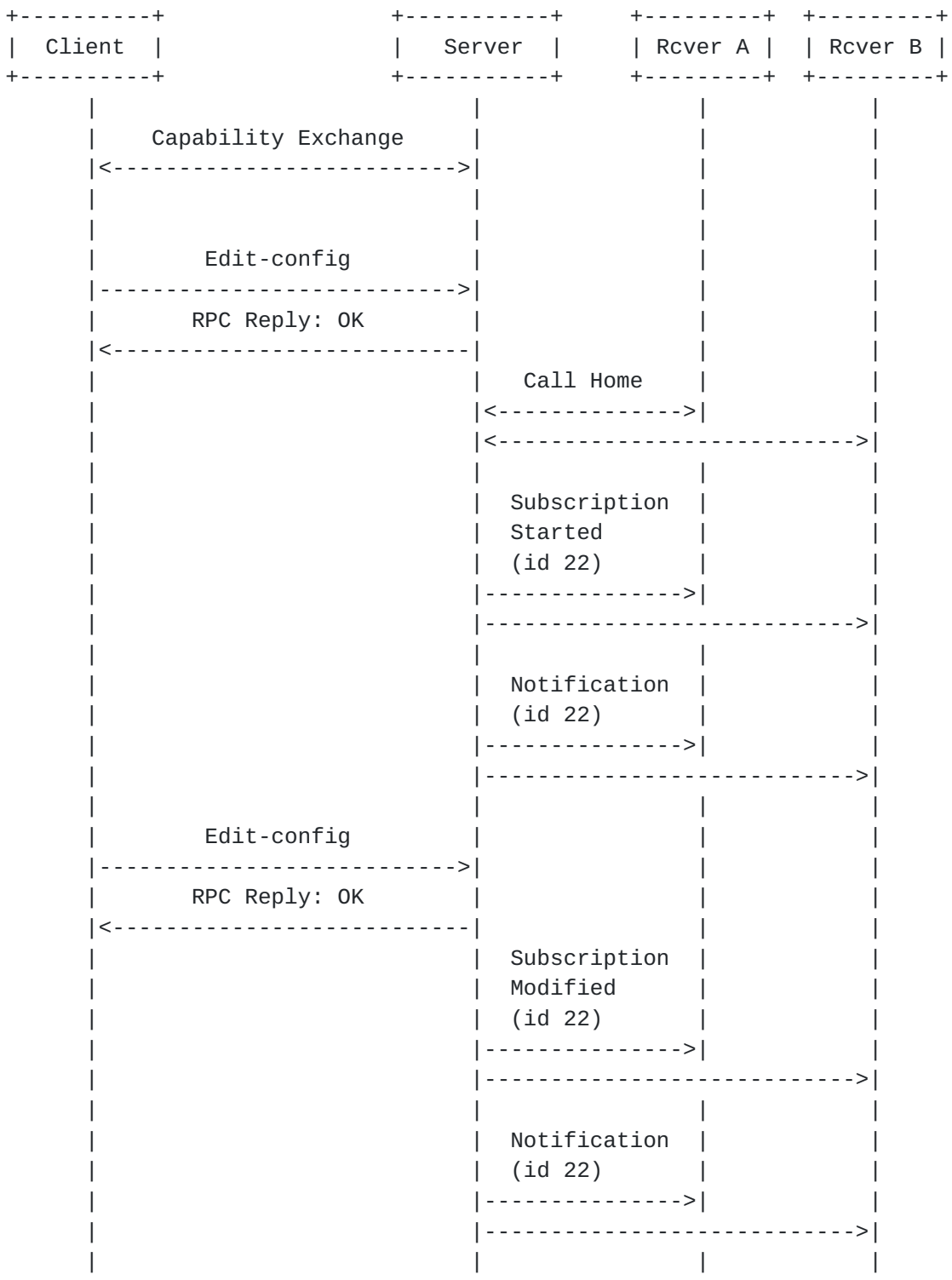

Figure 21: Message flow for subscription modification (configured subscription)

3.4.5. Deleting a Configured Subscription

Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example:

```
<rpc message-id="103"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
        xmlns:xc="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <subscription xc:operation="delete">
        <identifier>
          1922
        </identifier>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>

<rpc-reply message-id="103"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 22: Deleting a configured subscription

3.4.5.1. Message Flow Example

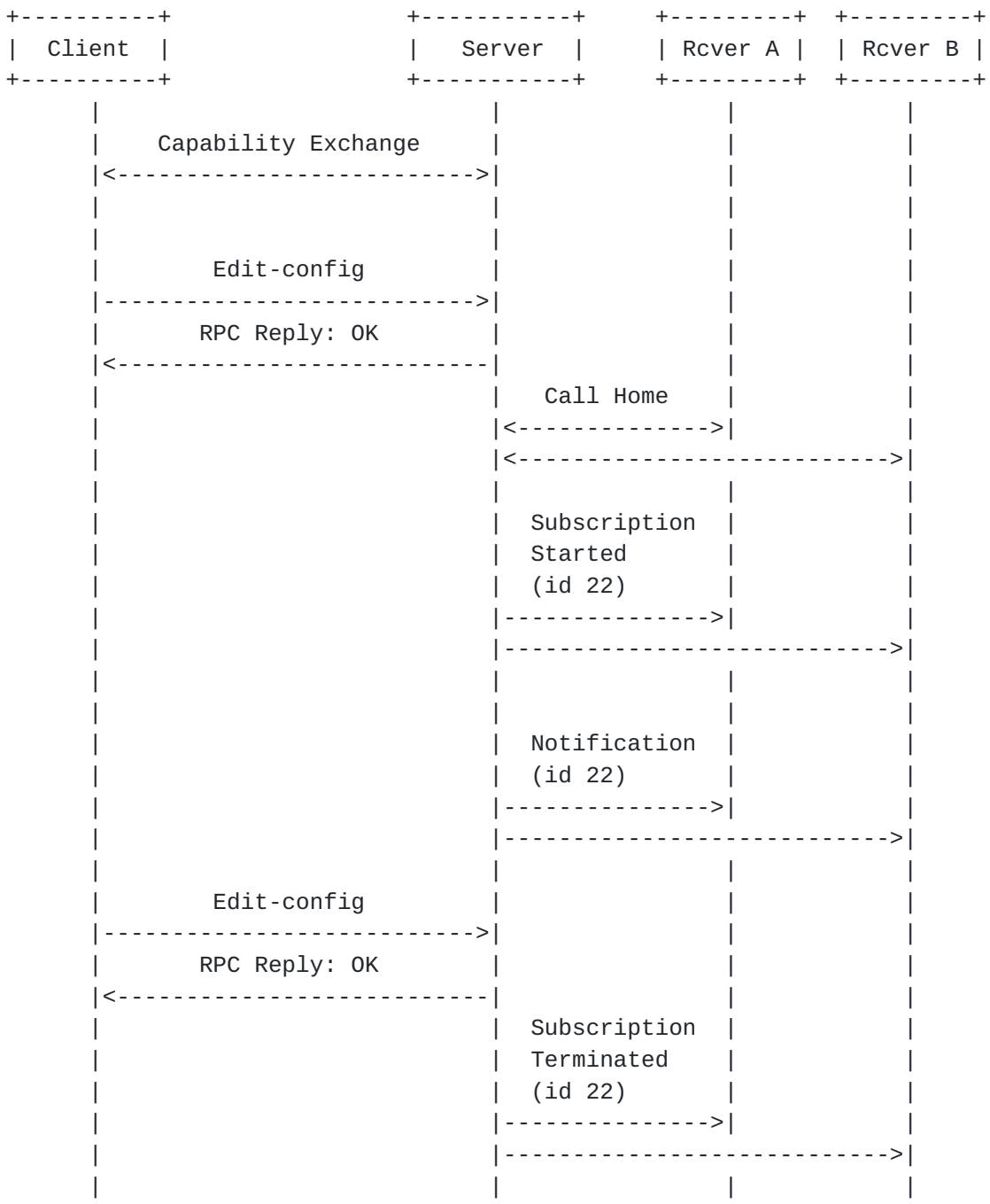

Figure 23: Message flow for subscription deletion (configured subscription)

3.4.6. Event (Data Plane) Notifications

Once a dynamic or configured subscription has been created, the NETCONF server sends (asynchronously) event notifications from the subscribed stream to receiver(s) over NETCONF. We refer to these as data plane notifications. The data model for Event Notifications is

defined in [subscribe]. This document extends that data model for supporting different encondigs.

The following is an example of an event notification from [RFC6020]:

```
notification link-failure {
  description "A link failure has been detected";
    leaf if-name {
      type leafref {
        path "/interface/name";
      }
    }
    leaf if-admin-status {
      type admin-status;
    }
    leaf if-oper-status {
      type oper-status;
    }
}
```

Figure 24: Definition of a event notification

This notification might result in the following, prior to it being placed into NETCONF. Note that the mandatory eventTime and Subscription id have been added.

```
<notification
       xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-id>39</subscription-id>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

Figure 25: Event notification

In order to support JSON encoding, this document extends the data model for Event Notifications, adding a new element, i.e., notification-contents-json. This element contains the event notification-specific tagged content in JSON. For the event notification above, the equivalent using JSON encoding would be

```
<notification
       xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-id>39</subscription-id>
  <notification-contents-json>
       {
          "acme-system:link-failure": {
            "if-name": "so-1/2/3.0",
            "if-admin-status": "up",
            "if-oper-status": "down"
          }
        }
  </notification-contents-json>
</notification>
```

Figure 26: Event notification using JSON encoding

3.4.7. Subscription State Notifications

In addition to data plane notifications, a publisher may send subscription state notifications (defined in [subscribe]) to indicate to receivers that an event related to the subscription management has occurred. Subscription state notifications cannot be filtered out. Next we exemplify them using both XML, and JSON encodings for the notification-specific content:

3.4.7.1. subscription-started and subscription-modified

A subscription-started would look like:

```
<notification
    xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-started
       xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0"/>
  <identifier>39</identifier>
  <event-filter-type>xpath</event-filter-type>
  <event-filter xmlns:ex="http://example.com/event/1.0"
            select="/ex:event[ex:eventClass='fault' and
                   (ex:severity='minor' or ex:severity='major'
                    or ex:severity='critical')]"/>
  </subscription-started/>
</notification>
```

Figure 27: subscription-started subscription state notification

The equivalent using JSON encoding would be:

```
<notification
       xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
       {
          "notif-bis:subscription-started": {
              "identifier" : 39
          }
        }
  </notification-contents-json>
</notification>
```

Figure 28: subscription-started subscription state notification (JSON)

The subscription-modified is identical, with just the word "started" being replaced by "modified".

3.4.7.2. notification-complete, subscription-resumed, and replay-complete

A notification-complete would look like:

```
<notification
    xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-complete
       xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <identifier>39</identifier>
  </notification-complete>
</notification>
```

Figure 29: notification-complete notification in XML

The equivalent using JSON encoding would be:

```
<notification
       xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
       {
          "netmod-notif:notification-complete": {
          "identifier" : 39
        }
  </notification-contents-json>
</notification>
```

Figure 30: notification-complete notification in JSON

The subscription-resumed and replay-complete are virutally identical, with "notification-complete" simply being replaced by "subscription-resumed" and "replay-complete" in both encodings.

3.4.7.3. subscription-terminated and subscription-suspended

A subscription-terminated would look like:

```
  <notification
      xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-terminated
         xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <identifier>39</identifier>
    <error-id>no-such-subscription</error-id>
    </subscription-terminated>
  </notification>
```

Figure 31: subscription-modified subscription state notification

The above, and the subscription-suspended are virutally identical, with "subscription-terminated" simply being replaced by "subscription-suspended".

3.4.7.4. Notification Message Flow Examples

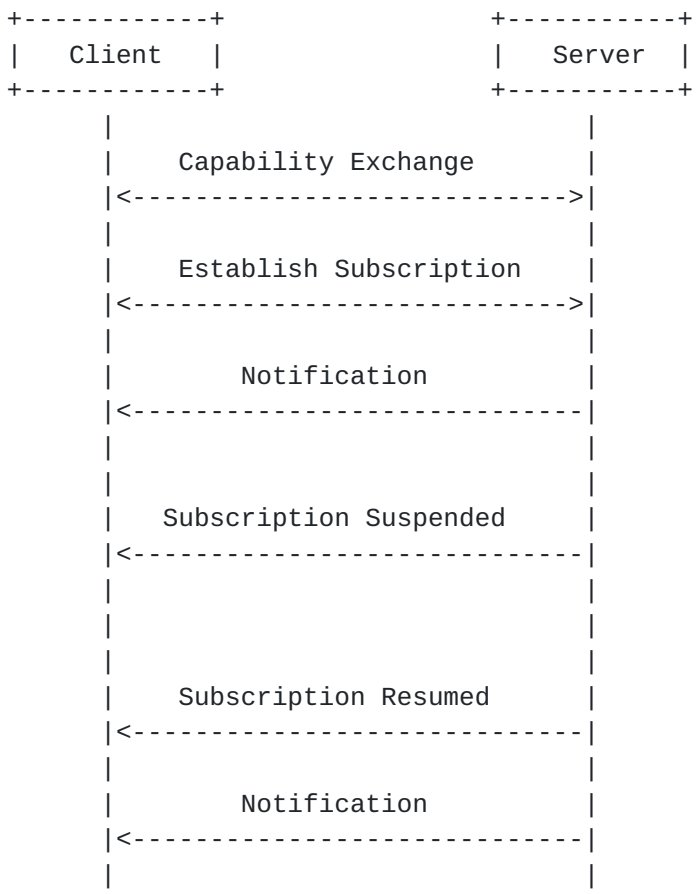

Figure 32: subscription-suspended and resumed notifications

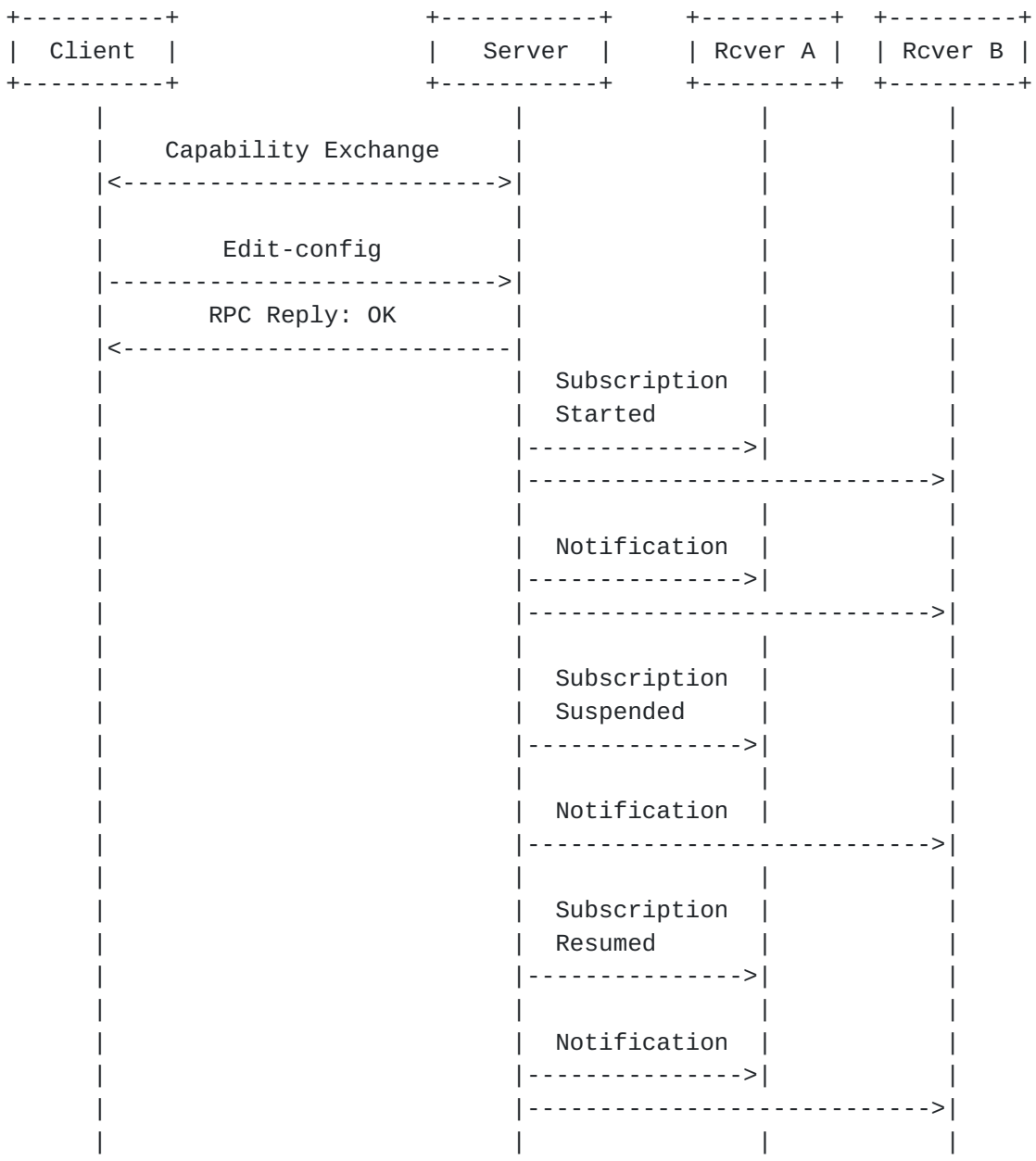

Figure 33: Suspended and resumed notificatiosn for a single configured receiver

4. Interleave Capability

The :interleave capability is originally defined in [RFC5277]. It is incorporated in this document with essentialy the same semantics. That is, the NETCONF server MUST receive, process, and respond to NETCONF requests on a session with active notification subscriptions. Note that subscription operations MUST be received, processed, and responded on a session with active notification subscriptions This mandatory requirement together with the :inteleave capability permits

a client performing all operations against a server using a single connection, allowing for better scalability with respect to the number of NETCONF sessions required to manage an entity. The :interleave capability is identified by the following string: urn:ietf:params:netconf:capability:interleave:1.0

5. Security Considerations

The <notification> elements are never sent before the transport layer and the NETCONF layer, including capabilities exchange, have been established and the manager has been identified and authenticated.

A secure transport must be used and the server must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of content involved.

The contents of notifications, as well as the names of event streams, may contain sensitive information and care should be taken to ensure that they are viewed only by authorized users. The NETCONF server MUST NOT include any content in a notification that the user is not authorized to view.

If a malicious or buggy NETCONF client sends a number of <establish-subscription>requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions MAY be terminated by terminating the suspect underlying NETCONF sessions. The server MAY also suspend or terminate a subset of the active subscriptions on the NETCONF session .

Configured subscriptions from one or more publishers could be used to overwhelm a receiver, which perhaps doesn't even support subscriptions. Clients that do not want pushed data need only terminate or refuse any transport sessions from the publisher.

The NETCONF Authorization Control Model [RFC6536] SHOULD be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user permissions to receive specific event notification types. The permissions are specified as a set of access control rules.

6. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Andy Bierman, Yan Gang, Sharon Chisholm, Hector Trevino, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, and Guangying Zheng.

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, <http://www.rfc-editor.org/info/rfc2119>.

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

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

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

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

7.2. Informative References

[subscribe] Voit, Eric., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscribing to Event Notifications", April 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>.

[yang-push] Clemm, A., Gonzalez Prieto, A., Voit, Eric., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", April 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Appendix A. Open Items

(To be removed by RFC editor prior to publication)

- Formal definition of: notification-contents-json. It depends on the formal definition of the notification element

- Specify how to indicate a stream is not part of the NETCONF stream. It depends on draft-ietf-netconf-subscribed-notifications

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

B.1. v03 to v04

- Added additional detail to "configured subscriptions"

- Added interleave capability

- Adjusted terminology to that in draft-ietf-netconf-subscribed-notifications

- Corrected namespaces in examples

B.2. v01 to v03

- Text simplifications throughout

- v02 had no meaningful changes

B.3. v00 to v01

- Added Call Home in solution for configured subscriptions.

- Clarified support for multiple subscription on a single session. No need to support multiple create-subscription.

- Added mapping between terminology in [yang-push] and [RFC6241] (the one followed in this document).

- Editorial improvements.

Authors' Addresses

Alberto Gonzalez Prieto
VMware

Email: agonzalezpri@vmware.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com