NETCONF
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2017

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
February 23, 2017

Subscribing to Event Notifications
draft-ietf-netconf-subscribed-notifications-00

Abstract

This document defines capabilities and operations for subscribing to content and providing asynchronous notification message delivery on that content. Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF and RESTCONF. The capabilities and operations defined in this document when using in conjunction with draft-ietf-netconf-netconf-event-notifications are intended to obsolete RFC 5277.

Status of This Memo

Copyright Notice

Table of Contents

```
   1.  Introduction . . . . . . . . . . . . . . . . . . . . . . .   3
     1.1.  Motivation . . . . . . . . . . . . . . . . . . . . . .   3
     1.2.  Terminology . . . . . . . . . . . . . . . . . . . . .   4
     1.3.  Solution Overview . . . . . . . . . . . . . . . . . .   5
   2.  Solution . . . . . . . . . . . . . . . . . . . . . . . . .   6
     2.1.  Event Streams . . . . . . . . . . . . . . . . . . . .   6
     2.2.  Filters . . . . . . . . . . . . . . . . . . . . . . .   6
     2.3.  Subscription State Model at the Publisher . . . . . . .   6
   3.  Data Model Trees for Event Notifications . . . . . . . . .   7
   4.  Dynamic Subscriptions . . . . . . . . . . . . . . . . . . .  11
     4.1.  Establishing a Subscription . . . . . . . . . . . . . .  11
     4.2.  Modifying a Subscription . . . . . . . . . . . . . . .  12
     4.3.  Deleting a Subscription . . . . . . . . . . . . . . . .  12
     4.4.  Killing a Subscription . . . . . . . . . . . . . . . .  13
   5.  Configured Subscriptions . . . . . . . . . . . . . . . . .  13
     5.1.  Establishing a Configured Subscription . . . . . . . .  14
     5.2.  Modifying a Configured Subscription . . . . . . . . . .  16
     5.3.  Deleting a Configured Subscription . . . . . . . . . .  16
   6.  Event (Data Plane) Notifications . . . . . . . . . . . . .  17
   7.  Subscription State Notifications . . . . . . . . . . . . .  18
     7.1.  subscription-started . . . . . . . . . . . . . . . . .  18
     7.2.  subscription-modified . . . . . . . . . . . . . . . . .  18
     7.3.  subscription-terminated . . . . . . . . . . . . . . . .  19
     7.4.  subscription-suspended . . . . . . . . . . . . . . . .  19
     7.5.  subscription-resumed . . . . . . . . . . . . . . . . .  19
     7.6.  notification-complete . . . . . . . . . . . . . . . . .  19
     7.7.  replay-complete . . . . . . . . . . . . . . . . . . . .  19
   8.  Administrative Functions . . . . . . . . . . . . . . . . .  20
     8.1.  Subscription Monitoring . . . . . . . . . . . . . . . .  20
     8.2.  Capability Advertisement . . . . . . . . . . . . . . .  20
     8.3.  Event Stream Discovery . . . . . . . . . . . . . . . .  21
   9.  Data Model for Event Notifications . . . . . . . . . . . .  21
   10. Considerations . . . . . . . . . . . . . . . . . . . . . .  41
     10.1.  Implementation Considerations . . . . . . . . . . . .  41
     10.2.  Security Considerations . . . . . . . . . . . . . . .  41
   11. Acknowledgments . . . . . . . . . . . . . . . . . . . . . .  42
   12. References . . . . . . . . . . . . . . . . . . . . . . . .  42
     12.1.  Normative References . . . . . . . . . . . . . . . . .  42
```

12.2. Informative References 43
Appendix A. Issues that are currently being worked and resolved 43
Appendix B. Changes between revisions 44
Authors' Addresses 45

1. Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service in a protocol-agnostic manner. This document defines capabilities and operations for providing asynchronous message notification delivery for notifications including those necessary to establish, monitor, and support subscriptions to notification delivery.

Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF [RFC6241] (defined in [I-D.ietf-netconf-netconf-event-notif]) and Restconf [RFC8040] (defined in [I-D.ietf-netconf-restconf-notif]). The capabilities and operations defined in this document are intended to obsolete RFC 5277, along with their mapping onto NETCONF transport.

1.1. Motivation

The motivation for this work is to enable the sending of transport agnostic asynchronous notification messages driven by a YANG Subscription that are consistent with the data model (content) and security model. Predating this work was [RFC5277] which defined a limited defines a notification mechanism for for NETCONF. However, there are various [RFC5277] has limitations, many of which have been exposed in [RFC7923].

The scope of the work aims at meeting the operational needs of network subscriptions:

o Ability to dynamically or statically subscribe to event notifications available on a publisher.

o Ability to negotiate acceptable dynamic subscription parameters.

o Ability to filter the subset of notifications to be pushed with stream-specific semantics.

o Ability for the notification payload to be interpreted independently of the transport protocol. (In other words, the encoded notification fully describes itself.)

o Mechanism to communicate the notifications.

- o Ability to replay locally logged notifications.

1.2. 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].

Configured subscription: A subscription installed via a configuration interface which persists across reboots.

Dynamic subscription: A subscription agreed between subscriber and publisher created via RPC subscription state signaling messages.

Event: An occurrence of something that may be of interest. (e.g., a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.)

Event notification: A set of information intended for a Receiver indicating that one or more Event(s) have occurred. Details of the Event(s) may be included within the Notification.

Filter: Evaluation criteria, which may be applied against a targeted set of objects/events in a subscription. Information traverses the filter only if specified filter criteria are met.

NACM: NETCONF Access Control Model.

OAM: Operations, Administration, Maintenance.

Publisher: An entity responsible for streaming event notifications per the terms of a Subscriptions

Receiver: A target to which a publisher pushes event notifications. For dynamic subscriptions, the receiver and subscriber will often be the same entity.

RPC: Remote Procedure Call.

Stream (also referred to as "event stream"): A continuous ordered set of events grouped under an explicit criteria.

Subscriber: An entity able to request and negotiate a contract for the receipt of event notifications from a publisher.

Subscription: A contract with a publisher, stipulating which information receiver(s) wishes to have pushed from the publisher without the need for further solicitation.

1.3. Solution Overview

This document describes mechanisms for subscribing and receiving event notifications from an event server publisher. This document has similarities to the capabilities orginally defined in [RFC5277]. This document extends the supported capabilties, and generalizes functionality to be protocol-agnostic.

Some enhancements over [RFC5277] include the ability to have multiple subscriptions on a single transport session, to terminate a single subscriptions without terminating the transport session, and to modify existing subscriptions.

The solution supports subscribing to event notifications using two mechanisms:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via RPC. If the publisher wants to serve this request, it will accept it, and then start pushing event notifications. If the publisher does not wish to serve it as requested, then an error response is returned. This response may include hints at subscription parameters which would have been accepted.

2. Configured subscriptions, which is an optional mechanism that enables managing subscriptions via a configuration interface so that a publisher can send event notifications to configured receiver(s).

Some key characteristics of configured and dynamic subscriptions include:

- o The lifetime of a dynamic subscription is limited by the lifetime of the subscriber session used to establish it. Typically loss of the transport session tears down any dependent dynamic subscriptions.

- o The lifetime of a configured subscription is driven by configuration being present on the running configuration. This implies configured subscriptions persist across reboots, and persists even when transport is unavailable.

- o Subscriptions can be modified or terminated at any point of their lifetime. Configured subscriptions can be modified by any configuration client with write rights on the configuration of the subscription.

Note that there is no mixing-and-matching of dynamic and configured subscriptions. Specifically, a configured subscription cannot be modified or deleted using RPC. Similarly, a subscription established via RPC cannot be modified through configuration operations.

The publisher may decide to terminate a dynamic subscription at any time. Similarly, it may decide to temporarily suspend the sending of event notifications for either configured or dynamic subscriptions. Such termination or suspension may be driven by the publisher running out of resources to serve the subscription, or by internal errors on the publisher.

2. Solution

2.1. Event Streams

An event stream is a set of events available for subscription from a publisher. It is out of the scope of this document to identify a) how streams are defined, b) how events are defined/generated, and c) how events are assigned to streams.

That said, there is one standardized event stream, this is the "NETCONF" event stream. The NETCONF event stream contains all NETCONF XML event information supported by the publisher, except for where it has been explicitly indicate that this info must be excluded from the NETCONF stream.

As events are raised by a system, they may be assigned to one or more streams. The event is distributed to receivers meeting all three critera: (1) a subscription includes the identified stream, (2) susbcription filtering allows the event to traverse, and (3) no access control rules prohibit the receiver from receiving the event.

2.2. Filters

a publisher implementation SHOULD support the ability to perform filtering of notification records per [RFC5277]. (TODO: since 5277 is to be obsoleted, we should describe the filter here.)

2.3. Subscription State Model at the Publisher

Below is the state machine of a subscription for the publisher. It is important to note that a subscription doesn’t exist at the publisher until it is accepted and made active. The mere request by a subscriber to establish a subscription is insufficient for that asserted subscription to be externally visible via this state machine.

```
                       .-------.
                       | start |
                       '-------'
                           |
                        establish
                           |
                           |   .----------modify--------------.
                           v   v                              '
                     .-----------.                          .-----------.
          .--------. |           |------>suspend------->|           |
        modify      '|  active   |                      | suspended |
          '--------->|           |<----resume----<------|           |
                     '-----------'                      '-----------'
                           |                                  |
                      delete/kill                        delete/kill
                           |                                  |
                           v                                  |
                       .-------.                              |
                       |  end  |<-----------------------------'
                       '-------'
```

Figure 1: Subscription states at publisher

Of interest in this state machine are the following:

o Successful <establish-subscription> or <modify-subscription> requests put the subscription into an active state.

o Failed <modify-subscription> requests will leave the subscription in its previous state, with no visible change to any streaming updates.

o A <delete-subscription> or <kill-subscription> will end the subscription.

3. Data Model Trees for Event Notifications

The YANG data model for event notifications is depicted in this section.

```
 module: ietf-subscribed-notifications
     +--ro streams
     |  +--ro stream*   stream
     +--rw filters
     |  +--rw filter* [identifier]
     |     +--rw identifier    filter-id
     |     +--rw (filter-type)?
     |        +--:(by-reference)
     |        |  +--rw filter-ref?   filter-ref
     |        +--:(event-filter)
     |           +--rw filter?
     +--rw subscription-config {configured-subscriptions}?
     |  +--rw subscription* [identifier]
     |     +--rw identifier          subscription-id
     |     +--rw stream?             stream
     |     +--rw encoding?           encoding
     |     +--rw stop-time?          yang:date-and-time
     |     +--rw (filter-type)?
     |     |  +--:(by-reference)
     |     |  |  +--rw filter-ref?         filter-ref
     |     |  +--:(event-filter)
     |     |     +--rw filter?
     |     +--rw receivers
     |     |  +--rw receiver* [address port]
     |     |     +--rw address     inet:host
     |     |     +--rw port        inet:port-number
     |     |     +--rw protocol?   transport-protocol
     |     +--rw (notification-origin)?
     |        +--:(interface-originated)
     |        |  +--rw source-interface?   if:interface-ref
     |        +--:(address-originated)
     |           +--rw source-vrf?         string
     |           +--rw source-address?     inet:ip-address-no-zone
     +--ro subscriptions
        +--ro subscription* [identifier]
           +--ro identifier                 subscription-id
           +--ro configured-subscription?
           |                      empty {configured-subscriptions}?
           +--ro stream?                    stream
           +--ro encoding?                  encoding
           +--ro replay-start-time?         yang:date-and-time
           +--ro stop-time?                 yang:date-and-time
           +--ro (filter-type)?
           |  +--:(by-reference)
           |  |  +--ro filter-ref?                filter-ref
           |  +--:(event-filter)
           |     +--ro filter?
           +--ro (notification-origin)?
           |  +--:(interface-originated)
           |  |  +--ro source-interface?          if:interface-ref
           |  +--:(address-originated)
           |     +--ro source-vrf?               string
           |     +--ro source-address?           inet:ip-address-no-zone
           +--ro receivers
           |  +--ro receiver* [address]
           |     +--ro address                   inet:host
           |     +--ro port                      inet:port-number
           |     +--ro protocol?                 transport-protocol
           |     +--ro pushed-notifications?     yang:counter64
           |     +--ro excluded-notifications?   yang:counter64
           +--ro subscription-status?       subscription-status

   rpcs:
     +---x establish-subscription
     |  +---w input
     |  |  +---w stream?              stream
     |  |  +---w encoding?            encoding
     |  |  +---w replay-start-time?   yang:date-and-time
     |  |  +---w stop-time?           yang:date-and-time
     |  |  +---w (filter-type)?
     |  |     +--:(by-reference)
     |  |     |  +---w filter-ref?          filter-ref
     |  |     +--:(event-filter)
     |  |        +---w filter?
     |  +--ro output
     |     +--ro subscription-result       subscription-result
     |     +--ro (result)?
     |        +--:(no-success)
     |        |  +--ro filter-failure?           string
     |        |  +--ro replay-start-time-hint?   yang:date-and-time
     |        +--:(success)
     |           +--ro identifier                subscription-id
     +---x modify-subscription
     |  +---w input
     |  |  +---w identifier?   subscription-id
     |  |  +---w stop-time?    yang:date-and-time
     |  |  +---w (filter-type)?
     |  |     +--:(by-reference)
     |  |     |  +---w filter-ref?   filter-ref
     |  |     +--:(event-filter)
     |  |        +---w filter?
     |  +--ro output
     |     +--ro subscription-result    subscription-result
     |     +--ro (result)?
     |        +--:(no-success)
     |           +--ro filter-failure?        string
     +---x delete-subscription
     |  +---w input
     |  |  +---w identifier    subscription-id
     |  +--ro output
     |     +--ro subscription-result    subscription-result
     +---x kill-subscription
        +---w input
        |  +---w identifier    subscription-id
        +--ro output
           +--ro subscription-result    subscription-result

   notifications:
     +---n replay-complete
     |  +--ro identifier    subscription-id
     +---n notification-complete
     |  +--ro identifier    subscription-id
     +---n subscription-started
     |  +--ro identifier           subscription-id
     |  +--ro stream?              stream
     |  +--ro encoding?            encoding
     |  +--ro replay-start-time?   yang:date-and-time
     |  +--ro stop-time?           yang:date-and-time
     |  +--ro (filter-type)?
     |     +--:(by-reference)
     |     |  +--ro filter-ref?          filter-ref
     |     +--:(event-filter)
     |        +--ro filter?
     +---n subscription-resumed
     |  +--ro identifier    subscription-id
     +---n subscription-modified
     |  +--ro identifier           subscription-id
     |  +--ro stream?              stream
     |  +--ro encoding?            encoding
     |  +--ro replay-start-time?   yang:date-and-time
     |  +--ro stop-time?           yang:date-and-time
     |  +--ro (filter-type)?
     |     +--:(by-reference)
     |     |  +--ro filter-ref?          filter-ref
     |     +--:(event-filter)
     |        +--ro filter?
     +---n subscription-terminated
     |  +--ro identifier        subscription-id
     |  +--ro error-id          subscription-errors
     |  +--ro filter-failure?   string
     +---n subscription-suspended
        +--ro identifier        subscription-id
        +--ro error-id          subscription-errors
        +--ro filter-failure?   string
```

The data model is structured as follows:

- o "Streams" contains a list of event streams that are supported by the publisher and that can be subscribed to.

- o "Filters" contains a configurable list of filters that can be applied to a subscription. This allows users to reference an existing filter definition as an alternative to defining a filter inline for each subscription.

- "Subscription-config" contains the configuration of configured subscriptions. The parameters of each configured subscription are a superset of the parameters of a dynamic subscription and use the same groupings. In addition, the configured subscriptions must also specify intended receivers and may specify the push source from which to send the stream of notification messages.

- "Subscriptions" contains a list of all subscriptions on a publisher, both configured and dynamic. It can be used to retrieve information about the subscriptions which an publisher is serving.

The data model also contains a number of notifications that allow a publisher to signal information about a subscription. Finally, the data model contains a number of RPC definitions that are used to manage dynamic subscriptions.

4. Dynamic Subscriptions

Dynamic subscriptions are managed via RPC.

4.1. Establishing a Subscription

The <establish-subscription> operation allows a subscriber to request the creation of a subscription via RPC.

The input parameters of the operation are:

- A filter which identifies what is being subscribed to, as well as what should be included (or not) in the pushed results.

- An optional stream which may identify or reduce the domain of events against which the subscription is applied.

- The desired encoding for the returned events. By default, updates are encoded using XML. Other encodings may be supported, such as JSON.

- An optional stop time for the subscription.

- An optional start time which indicates that this subscription is requesting a replay push of events previously generated.

If the publisher cannot satisfy the <establish-subscription> request, it sends a negative <subscription-result> element. If the subscriber has no authorization to establish the subscription, the <subscription-result> indicates an authorization error. Optionally, the <subscription-result> may include one or more hints on alternative input parameters and value which would have resulted in an accepted subscription.

Subscription requests must fail if a filter with invalid syntax is provided or if the name of a non-existent stream is provided.

4.1.1. Replay Subscription

The presence of a start time indicates that this is a replay subscription. The start time must be earlier than the current time. If the start time points earlier than the maintained history of Publisher’s event buffer, then the subscription must be rejected. In this case the error response to the <establish-subscription> request should include a start time supportable by the Publisher.

4.2. Modifying a Subscription

The <modify-subscription> operation permits changing the terms of an existing dynamic subscription previously established on that transport session. Subscriptions created by configuration operations cannot be modified via this RPC. Dynamic subscriptions can be modified one or multiple times. If the publisher accepts the requested modifications, it immediately starts sending events based on the new terms, completely ignoring the previous ones. If the publisher rejects the request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. The contents of a such a rejected modification may include one or more hints on alternative input parameters and value which would have resulted in a successfully modified subscription.

Dynamic subscriptions established via RPC can only be modified (or deleted) via RPC using the same transport session used to establish that subscription.

4.3. Deleting a Subscription

The <delete-subscription> operation permits canceling an existing subscription previously established on that transport session. If the publisher accepts the request, it immediately stops sending events for the subscription. If the publisher rejects the request, all subscriptions remain as prior to the request. That is, the request has no impact whatsoever.

Subscriptions established via RPC can only be deleted via RPC using the same transport session used for subscription establishment.

Configured subscriptions cannot be deleted using RPCs. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to delete configured subscriptions.

4.4. Killing a Subscription

The <kill-subscription> operation permits an operator to end any dynamic subscription. The publisher must accept the request for any dynamic subscription, and immediately stop sending events.

Configured subscriptions cannot be kill using this RPC. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to kill a configured subscription.

5. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface.

Configured subscriptions persist across reboots, and persist even when transport is unavailable.

Configured subscriptions can be modified by any configuration client with write permissions for the configuration of the subscription. Subscriptions can be modified or terminated via the configuration interface at any point of their lifetime.

Supporting configured subscriptions is optional and advertised using the "configured-subscriptions" feature.

In addition to subscription parameters that apply to dynamic subscriptions, the following additional parameters apply to configured subscriptions:

- One or more receiver IP addresses (and corresponding ports)intended as the destination for push updates for each subscription. In addition, the transport protocol for each destination may be defined.

- Optional parameters to identify an egress interface or IP address / VRF where a subscription updates should be pushed from the publisher. If not included, push updates will go off a default interface for the device.

5.1. Establishing a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level subtree subscription-config. There are two key differences between RPC and <edit-config> RPC operations for subscription establishment. Firstly, <edit-config> operations install a subscription without question, while RPCs may support negotiation and rejection of requests. Secondly, while RPCs mandate that the subscriber establishing the subscription is the only receiver of the notifications, <edit-config> operations permit specifying receivers independent of any tracked subscriber. Immediately after a subscription is successfully established, the publisher sends to any newly active receivers a control-plane notification stating the subscription has been established (subscription-started).

Because there is no explicit association with an existing transport session, <edit-config> operations require additional parameters to indicate the receivers of the notifications and possibly the source of the notifications such as a specific egress interface.

For example at subscription establishment if NETCONF transport is being used, a 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>
                   <subscription-id>
                       1922
                   </subscription-id>
                   <stream>
                       foo
                   </stream>
                   <receiver>
                       <address>
                           1.2.3.4
                       </address>
                       <port>
                           1234
                       </port>
                   </receiver>
               </subscription>
           </subscription-config>
       </edit-config>
   </rpc>
```

Figure 2: Configured subscription creation via NETCONF

if the request is accepted, the publisher would reply:

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

Figure 3: Successful NETCONF configured subscription response

if the request is not accepted because the publisher cannot serve it, the publisher 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</error-severity>
        <error-message xml:lang="en">
            Temporarily the publisher cannot serve this
            subscription due to the current workload.
        </error-message>
    </rpc-error>
</rpc-reply>
```

Figure 4: A NETCONF response for a failed configured subscription creation

5.2. Modifying a Configured Subscription

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

Immediately after a subscription is successfully modified, the publisher sends to the existing receivers a control-plane notification stating the subscription has been modified (i.e., subscription-modified).

If the modification involved adding and/or removing receivers, those modified receivers are sent control-plane notifications, indicating they have been added (i.e, subscription-started to a specific receiver) or removed (i.e., subscription-terminated to a specific receiver.)

5.3. Deleting a Configured Subscription

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

```
DELETE /subscription-config/subscription=1922 HTTP/1.1
Host: example.com

HTTP/1.1 204 No Content
Date: Sun, 24 Jul 2016 11:23:40 GMT
Server: example-server
```

Figure 5: Deleting a configured subscription

Immediately after a subscription is successfully deleted, the publisher sends to all receivers of that subscription a control-plane notification stating the subscription has been terminated (subscription-terminated).

6. Event (Data Plane) Notifications

Once a subscription has been set up, the publisher streams (asynchronously) notifications per the terms of the subscription. We refer to these as event notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the session used to establish the subscription. For configured subscriptions, event notifications are sent over the specified connections.

An event notification is sent to a receiver when something of interest occurs which is able to traverse all specified filtering and access control criteria. The event notification must include:

- o a subscription-id element of type uint32 which corresponds to responsible subscription in the Publisher.

- o an eventTime element which provides the time the event was generated by the event source. This event time parameter is of type dateTime and compliant to [RFC3339]. Implementations must support time zones.

- o the event notification content tagged and provided by a source in the publisher.

The following is an example of a compliant event notification. This example extending the example within [RFC7950] section 7.16.3 to include the mandatory information described above:

```
<notification
       xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
        <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-id>500</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 6: Data plane notification

While this extended [RFC7950] section 7.16 notification provides a valid method of encapsulating subscribed notifications, other transport encapsulation methods are also viable. Improvements may be achieved in some implementations in the following ways:

- o transport efficiency may be gained by allowing the encapsulation and bundled push of multiple events within the same event notification.

- o identifiers to designate the current and previous event notification can be used to discover duplicated and dropped notifications

- o additional header types can be used to pass relevant metadata.

- o a signature or hash can be included to verify the efficacy of the Publisher

This is being explored in NETMOD Notifications 2.0 [I-D.voit-notifications2].

7. Subscription State Notifications

In addition to data plane notifications, a publisher may send subscription state notifications to indicate to receivers that an event related to the subscription management has occurred.

Subscription state notifications are unlike other notifications in that they are not general-purpose notifications. They cannot be filtered out, and they are delivered only to directly impacted receiver(s) of a subscription. The definition of subscription state notifications is distinct from other notifications by making use of a YANG extension tagging them as subscription state notification.

Subscription state notifications include indications that a replay of notifications has been completed, that a subscription is done sending notifications because an end time has been reached, and that a subscription has started, been modified, been terminated, or been suspended. They are described in the following subsections.

7.1. subscription-started

This notification indicates that a configured subscription has started and data updates are beginning to be sent. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

7.2. subscription-modified

This notification indicates that a configured subscription has been modified successfully. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

7.3. subscription-terminated

This notification indicates that a subscription has been terminated by the publisher. The notification includes the reason for the termination. The publisher may decide to terminate a subscription when it is running out of resources for serving it, an internal error occurs, etc. Publisher-driven terminations are notified to all receivers. The management plane can also terminate configured subscriptions using configuration operations.

Subscribers can terminate via RPC subscriptions established via a delete-subscription RPC. In such cases, no subscription-terminated notifications are sent. However if a kill-subscription RPC is sent, or some other event results in the end of a susbcription, then there must be a notification that the subscription has been ended.

7.4. subscription-suspended

This notification indicates that a publisher has suspended a subscription. The notification includes the reason for the suspension. A possible reason is the lack of resources to serve it. No further data plane notifications will be sent until the subscription resumes. Suspensions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

7.5. subscription-resumed

This notification indicates that a previously suspended dubscription has been resumed. Data plane notifications generated in the future will be sent after the subscription terms. Resumptions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

7.6. notification-complete

This notification is sent to indicate that a subscription, which includes a stop time, has finished passing events.

7.7. replay-complete

This notification indicates that all of the notifications prior to the current time have been sent. This includes new notifications generated since the start of the subscription. This notification must not be sent for any other reason.

If subscription contains no stop time, or has a stop time which has not been reached, then after the replay-complete notification has been sent notifications will be sent in sequence as they arise naturally within the system.

8. Administrative Functions

8.1. Subscription Monitoring

Container "subscriptions" in the YANG module below contains the state of all subscriptions that are currently active. This includes subscriptions that were established (and have not yet been deleted) using RPCs, as well as subscriptions that have been configured as part of configuration. Using the <get> operation with NETCONF, or subscribing to this information via [I-D.ietf-netconf-yang-push] allows the status of subscriptions to be monitored.

Each subscription is represented as a list element. The associated information includes an identifier for the subscription, a subscription status, as well as the various subscription parameters that are in effect. The subscription status indicates whether the subscription is currently active and healthy, or if it is degraded in some form. Leaf "configured-subscription" indicates whether the subscription came into being via configuration or via RPC.

Subscriptions that were established by RPC are removed from the list once they expire (reaching stop-time) or when they are terminated. Subscriptions that were established by configuration need to be deleted from the configuration by a configuration editing operation even if the stop time has been passed.

8.2. Capability Advertisement

Capabilities are advertised in messages sent by each peer during session establishment [RFC6241]. Publishers supporting the features in this document must advertise the capability "urn:ietf:params:netconf:capability:notification:2.0".

The mechanism defined in this document is identified by "urn:ietf:params:netconf:capability:notification:2.0". If a subscriber only supports [RFC5277] and not this specification, then they will recognize the capability "urn:ietf:params:netconf:capability:notification:1.0" and ignore the capability defined in this document.

8.3. Event Stream Discovery

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. A client can retrieve this list like any other YANG-defined data, for example using the <get> operation when using NETCONF.

9. Data Model for Event Notifications

```
<CODE BEGINS> file "ietf-subscribed-notifications.yang"
module ietf-subscribed-notifications {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications";

  prefix sn;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import ietf-interfaces {
    prefix if;
  }

  organization "IETF";
  contact
    "WG Web:   &lt;http://tools.ietf.org/wg/netconf/&gt;
     WG List:  &lt;mailto:netconf@ietf.org&gt;

     WG Chair: Mahesh Jethanandani
               &lt;mailto:mjethanandani@gmail.com&gt;

     WG Chair: Mehmet Ersue
               &lt;mailto:mehmet.ersue@nokia.com&gt;

     Editor:   Alexander Clemm
               &lt;mailto:ludwig@clemm.org&gt;

     Editor:   Eric Voit
               &lt;mailto:evoit@cisco.com&gt;

     Editor:   Alberto Gonzalez Prieto
               &lt;mailto:albertgo@cisco.com&gt;

     Editor:   Einar Nilsen-Nygaard
```

```
                &lt;mailto:einarnn@cisco.com&gt;

       Editor:   Ambika Prasad Tripathy
                 &lt;mailto:ambtripa@cisco.com&gt";


     description
       "This module contains conceptual YANG specifications for NETCONF
       Event Notifications.";

     revision 2017-02-23 {
       description
         "Tweaks to remove two notifications, RPC for create subscription
         refined with stream default, new grouping to eliminate some
         dymanically modifiable parameters in modifiy subscription RPC";
       reference
         "draft-ietf-netconf-subscribed-notifications-00";
     }


     /*
      * FEATURES
      */

     feature json {
       description
         "This feature indicates that JSON encoding of notifications
          is supported.";
     }

     feature configured-subscriptions {
       description
         "This feature indicates that management plane configuration
          of subscription is supported.";
     }

     /*
      * EXTENSIONS
      */

     extension subscription-state-notif {
       description
         "This statement applies only to notifications. It indicates that
          the notification is a subscription state notification (aka OAM
          notification). Therefore it does not participate in a regular
          event stream and does not need to be specifically subscribed
          in order to receive notifications.";
     }
```

```
  /*
   * IDENTITIES
   */

   /* Identities for streams */
   identity stream {
    description
      "Base identity to represent a generic stream of event
       notifications.";
  }

  identity NETCONF {
    base stream;
    description
      "Default NETCONF event stream, containing events based on
       notifications defined as YANG modules that are supported by the
       system.  This contains the same set of events in a default
       RFC-5277 NETCONF stream";
  }

  /* Identities for subscription results */
  identity subscription-result {
    description
      "Base identity for RPC responses to requests surrounding
       management (e.g. creation, modification, deletion) of
       subscriptions.";
  }

  identity ok {
    base subscription-result;
    description
      "OK - RPC was successful and was performed as requested.";
  }

  identity error {
    base subscription-result;
    description
      "RPC was not successful.
       Base identity for error return codes.";
  }

  /* Identities for subscription stream status */
  identity subscription-stream-status {
    description
      "Base identity for the status of subscriptions and datastreams.";
  }

  identity active {
    base subscription-stream-status;
    description
      "Status is active and healthy.";
  }

  identity inactive {
    base subscription-stream-status;
    description
      "Status is inactive, for example outside the interval between
      start time and stop time.";
  }

  identity suspended {
    base subscription-stream-status;
    description
      "The status is suspended, meaning that the publisher is currently
       unable to provide the negotiated updates for the subscription.";
  }

  identity in-error {
    base subscription-stream-status;
    description
      "The status is in error or degraded, meaning that stream and/or
      subscription is currently unable to provide the negotiated
      notifications.";
  }

  /* Identities for subscription errors */

  identity internal-error {
    base error;
    description
      "Error within publisher prohibits operation.";
  }

  identity suspension-timeout {
    base error;
    description
     "Termination of previously suspended subscription. The publisher
      has eliminated the subscription as it exceeded a time limit for
      suspension.";
  }

  identity stream-unavailable {
    base error;
    description
     "Stream name does not exist or is not available to the receiver.";
  }
```

```
  identity encoding-unavailable {
    base error;
    description
     "Encoding not supported";
  }

  identity replay-unsupported {
    base error;
    description
     "Replay cannot be performed for this subscription. The publisher
     does not provide the requested historic information via replay.";
  }

  identity history-unavailable {
    base error;
    description
     "Replay request too far into the past. The publisher does store
      historic information for all parts of requested subscription, but
      not back to the requested timestamp.";
  }

  identity filter-unavailable {
    base error;
    description
     "Referenced filter does not exist";
  }

  identity filter-unsupported {
    base error;
    description
     "Cannot parse syntax within the filter. Failure can be from a
      syntax error, or a syntax too complex to be processed by the
      platform. The supplemental info should include the invalid part
      of the filter.";
  }

  identity namespace-unavailable {
    base error;
    description
     "Referenced namespace doesn’t exist or is unavailable
      to the receiver.";
  }

  identity no-such-subscription {
    base error;
    description
     "Referenced subscription doesn’t exist. This may be as a result of
      a non-existent subscription ID, an ID which belongs to another
```

```
        subscriber, or an ID for acceptable subscription which has been
        statically configured.";
    }

    /* Identities for encodings */
    identity encodings {
      description
        "Base identity to represent data encodings";
    }

    identity encode-xml {
      base encodings;
      description
        "Encode data using XML";
    }

    identity encode-json {
      base encodings;
      description
        "Encode data using JSON";
    }

    /* Identities for transports */
    identity transport {
      description
        "An identity that represents a transport protocol for event
         notifications";
    }

    identity netconf {
      base transport;
      description
        "Netconf notifications as a transport.";
    }

    /*
     * TYPEDEFs
     */

    typedef subscription-id {
      type uint32;
      description
        "A type for subscription identifiers.";
    }

    typedef filter-id {
      type uint32;
      description
        "A type to identify filters which can be associated with a
         subscription.";
    }

    typedef subscription-result {
      type identityref {
        base subscription-result;
      }
      description
        "The result of a subscription operation";
    }

    typedef subscription-errors {
      type identityref {
        base error;
      }
      description
        "The reason for the failure of an RPC request or the sending of a
         subscription suspension or termination notification";
    }

    typedef encoding {
      type identityref {
        base encodings;
      }
      description
        "Specifies a data encoding, e.g. for a data subscription.";
    }

    typedef subscription-status {
      type identityref {
        base subscription-stream-status;
      }
      description
        "Specifies the status of a subscription or datastream.";
    }

    typedef transport-protocol {
      type identityref {
        base transport;
      }
      description
        "Specifies transport protocol used to send notifications to a
         receiver.";
    }

    typedef notification-origin {
      type enumeration {
        enum "interface-originated" {
          description
            "Notifications will be sent from a specific interface on a
             publisher";
        }
        enum "address-originated" {
          description
            "Notifications will be sent from a specific address on a
             publisher";
        }
      }
      description
        "Specifies from where notifications will be sourced when
         being sent by the publisher.";
    }

    typedef stream {
      type identityref {
        base stream;
      }
      description
        "Specifies a system-provided datastream.";
    }

    typedef filter-ref {
      type leafref {
        path "/sn:filters/sn:filter/sn:identifier";
      }
      description
        "This type is used to reference a filter.";
    }

    /*
     * GROUPINGS
     */

    grouping base-filter {
      description
        "This grouping defines the base for filters for notification
        events.";
      choice filter-type {
        description
          "A filter needs to be a single filter of a given type.  Mixing
          and matching of multiple filters does not occur at the level of
          this grouping.";
        case by-reference {
          description
            "Incorporate a filter that has been configured separately.";
          leaf filter-ref {
            type filter-ref;
            description
              "References an existing filter which is to be applied to
              the potential events of the subscription.";
          }
        }
        case event-filter {
          anyxml filter {
            description
             "Filter which excludes whole event-notifications. If a filter
              element is specified to look for data of a particular
              value, and the data item is not present within a particular
              event notification for its value to be checked against, the
              notification will be filtered  out. For example, if one
              were to check for 'severity=critical' in a configuration
              event notification where this field was not supported, then
              the notification would be filtered out. For subtree
              filtering, a non-empty node set means that the filter
              matches.  For XPath filtering, the mechanisms defined in
              [XPATH] should be used to convert the returned  value to
              boolean.";
          }
         }
       }
    }

    grouping subscription-policy-non-configurable {
      description
        "This grouping describes the information which can only be set
         in a dynamic subscription request via RPC.";
      leaf replay-start-time {
        type yang:date-and-time;
        description
          "Used to trigger the replay feature and indicate that the
          replay should start at the time specified.  If replay-start-time
          is not present, this is not a replay subscription and event
          pushes should start immediately.  It is never valid to
          specify start times that are later than or equal to the
          current time.";
      }
    }

    grouping subscription-policy-non-modifiable {
      description
        "This grouping describes the information in a subscription which
        should not change during the life of the subscription.";
      leaf stream {
        type stream;
        description
          "Indicates which stream of events is of interest.
           If not present, events in the default NETCONF stream
           will be sent.";
      }
      leaf encoding {
        type encoding;
        default "encode-xml";
        description
          "The type of encoding for the subscribed data.
           Default is XML";
      }
    }

    grouping subscription-policy-modifiable {
      description
        "This grouping describes all objects which may be changed
        in a subscription via an RPC.";
      leaf stop-time {
        type yang:date-and-time;
        description
          "Identifies a time after which notification events should not
          be sent.  If stop-time is not present, the notifications will
          continue until the subscription is terminated.  If
          replay-start-time exists, stop-time must for a subsequent time.
          If replay-start-time doesn't exist, stop-time must for a future
          time.";
      }
      uses base-filter;
    }

    grouping subscription-policy {
      description
        "This grouping describes information concerning a subscription.";
      uses subscription-policy-non-modifiable;
      uses subscription-policy-non-configurable;
      uses subscription-policy-modifiable;
    }

    grouping notification-origin-info {
      description
        "Defines the sender source from which notifications for a
         configured subscription are sent.";
      choice notification-origin {
        description
          "Identifies the egress interface on the Publisher from which
           notifications will or are being sent.";
        case interface-originated {
          description
            "When the push source is out of an interface on the
             Publisher established via static configuration.";
          leaf source-interface {
            type if:interface-ref;
            description
              "References the interface for notifications.";
          }
        }
        case address-originated {
          description
            "When the push source is out of an IP address on the
             Publisher established via static configuration.";
          leaf source-vrf {
            type string;
            description
              "Network instance name for the VRF.  This could also have
              been a leafref to draft-ietf-rtgwg-ni-model, but that model
              in not complete, and may not be implemented on a box.";
          }
          leaf source-address {
            type inet:ip-address-no-zone;
            description
              "The source address for the notifications.";
          }
        }
      }
    }

    grouping receiver-info {
      description
        "Defines where and how to get notifications for a configured
        subscriptions to one or more targeted recipient.  This includes
        specifying the destination addressing as well as a transport
        protocol acceptable to the reciever.";
      container receivers {
        description
          "Set of receivers in a subscription.";
        list receiver {
          key "address port";
          min-elements 1;
          description
            "A single host or multipoint address intended as a target
             for the notifications for a subscription.";
          leaf address {
            type inet:host;
            mandatory true;
            description
              "Specifies the address for the traffic to reach a remote
              host. One of the following must be specified: an ipv4
              address, an ipv6 address, or a host name.";
          }
          leaf port {
            type inet:port-number;
            mandatory true;
            description
              "This leaf specifies the port number to use for messages
               destined for a receiver.";
          }
          leaf protocol {
            type transport-protocol;
            default "netconf";
            description
              "This leaf specifies the transport protocol used
               to deliver messages destined for the receiver.  Each
               protocol may use the address and port information
               differently as applicable.";
          }
        }
      }
    }

    grouping error-identifier {
      description
        "A code passed back within an RPC response to describe why the RFC
         has failed, or within a state change notification to describe why
         the change has occurred.";
      leaf error-id {
        type subscription-errors;
        mandatory true;
        description
          "Identifies the subscription error condition.";
      }
    }

    grouping error-hints {
      description
        "Objects passed back within an RPC response to descibe why the RFC
         has failed, or within a state change notification to describe why
         the change has occured.";
      leaf filter-failure {
        type string;
        description
          "Information describing where and/or why a provided filter was
          unsupportable for a subscription.";
      }
    }

    grouping subscription-response-with-hints {
      description
        "Defines the output for the establish-subscription and
         modify-subscription RPCs.";
      leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
          "Indicates whether subscription is operational, or if a problem
           was encountered.";
      }
      choice result {
        description
          "Depending on the subscription result, different data is
           returned.";
        case no-success {
          description
            "This case applies when a subscription request was not
             successful and no subscription was created (or modified) as a
             result.  In this case, information MAY be returned that
             indicates suggested parameter settings that would have a
             high likelihood of succeeding in a subsequent establish-
             subscription or modify-subscription request.";
          uses error-hints;
        }
      }
    }


    /*
     * RPCs
     */

    rpc establish-subscription {
      description
        "This RPC allows a subscriber to create (and possibly negotiate)
         a subscription on its own behalf.  If successful, the
         subscription remains in effect for the duration of the
         subscriber’s association with the publisher, or until the
         subscription is terminated. In case an error (as indicated by
         subscription-result) is returned, the subscription is not
         created.  In that case, the RPC output MAY include suggested
         parameter settings that would have a high likelihood of
         succeeding in a subsequent establish-subscription request.";
      input {
        uses subscription-policy;
      }
      output {
        uses subscription-response-with-hints  {
          augment "result" {
            description
              "Allows information to be passed back as part of a
               successful subscription establishment.";
            case success {
              description
                "This case is used when the subscription request was
                 successful.";
              leaf identifier {
                type subscription-id;
                mandatory true;
                description
                  "Identifier used for this subscription.";
              }
            }
          }
          augment "result/no-success" {
            description
              "Contains establish RPC specific objects which can be
               returned as hints for future attempts.";
            leaf replay-start-time-hint {
              type yang:date-and-time;
              description
                "If a replay has been requested, but the requested replay
                   time cannot be honored, this may provide a hint at an
                 alternate time which may be supportable.";
            }
          }
        }
      }
    }

    rpc modify-subscription {
      description
        "This RPC allows a subscriber to modify a subscription that was
         previously created using establish-subscription.  If successful,
         the changed subscription remains in effect for the duration of
         the subscriber’s association with the publisher, or until the
         subscription is again modified or terminated.  In case an error
         is returned (as indicated by subscription-result), the
         subscription is not modified and the original subscription
         parameters remain in effect.  In that case, the rpc error
         response MAY include suggested parameter hints that would have
         a high likelihood of succeeding in a subsequent
```

```
          modify-subscription request.";
       input {
         leaf identifier {
           type subscription-id;
           description
             "Identifier to use for this subscription.";
         }
         uses subscription-policy-modifiable;
       }
       output {
         uses subscription-response-with-hints;
       }
     }

     rpc delete-subscription {
       description
         "This RPC allows a subscriber to delete a subscription that
          was previously created from by that same subscriber using the
          establish-subscription RPC.";
       input {
         leaf identifier {
           type subscription-id;
           mandatory true;
           description
             "Identifier of the subscription that is to be deleted.
              Only subscriptions that were created using
              establish-subscription can be deleted via this RPC.";
         }
       }
       output {
         leaf subscription-result {
           type subscription-result;
           mandatory true;
           description
             "Indicates whether subscription is operational, or if a
             problem was encountered.";
         }
       }
     }

     rpc kill-subscription {
       description
         "This RPC allows an operator to delete a dynamic subscription
         without restrictions on the originating subscriber or underlying
         transport session.";
       input {
         leaf identifier {
           type subscription-id;
```

```
        mandatory true;
        description
          "Identifier of the subscription that is to be deleted. Only
            subscriptions that were created using establish-subscription
           can be deleted via this RPC.";
      }
    }
    output {
      leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
          "Indicates whether subscription is operational, or if a
           problem was encountered.";
      }
    }
  }

  /*
   * NOTIFICATIONS
   */

  notification replay-complete {
    sn:subscription-state-notif;
    description
      "This notification is sent to indicate that all of the replay
        notifications have been sent. It must not be sent for any other
       reason.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }

  notification notification-complete {
    sn:subscription-state-notif;
    description
      "This notification is sent to indicate that a subscription, has
       finished passing events.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }
  notification subscription-started {
    sn:subscription-state-notif;
    description
      "This notification indicates that a subscription has started and
        notifications are beginning to be sent. This notification shall
       only be sent to receivers of a subscription; it does not
       constitute a general-purpose notification.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses subscription-policy;
  }

  notification subscription-resumed {
    sn:subscription-state-notif;
    description
      "This notification indicates that a subscription that had
       previously been suspended has resumed. Notifications will once
       again be sent.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }

  notification subscription-modified {
    sn:subscription-state-notif;
    description
      "This notification indicates that a subscription has been
       modified.  Notifications sent from this point on will conform to
       the modified terms of the subscription.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses subscription-policy;
  }

  notification subscription-terminated {
    sn:subscription-state-notif;
    description
```

```
      "This notification indicates that a subscription has been
       terminated.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses error-identifier;
    uses error-hints;
  }

  notification subscription-suspended {
    sn:subscription-state-notif;
    description
      "This notification indicates that a suspension of the
       subscription by the publisher has occurred.  No further
       notifications will be sent until the subscription resumes.
       This notification shall only be sent to receivers of a
       subscription; it does not constitute a general-purpose
       notification.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses error-identifier;
    uses error-hints;
  }

  /*
   * DATA NODES
   */

  container streams {
    config false;
    description
      "This container contains a leaf list of built-in
       streams that are provided by the system.";
    leaf-list stream {
      type stream;
      description
        "Identifies the built-in streams that are supported by the
         system.  Built-in streams are associated with their own
         identities, each of which carries a special semantics.
         In case configurable custom streams are supported,
         as indicated by the custom-stream identity, the configuration
         of those custom streams is provided separately.";
    }
  }
  container filters {
    description
      "This container contains a list of configurable filters
       that can be applied to subscriptions.  This facilitates
       the reuse of complex filters once defined.";
    list filter {
      key "identifier";
      description
        "A list of configurable filters that can be applied to
         subscriptions.";
      leaf identifier {
        type filter-id;
        description
          "An identifier to differentiate between filters.";
      }
      uses base-filter;
    }
  }
  container subscription-config {
    if-feature "configured-subscriptions";
    description
      "Contains the list of subscriptions that are configured,
       as opposed to established via RPC or other means.";
    list subscription {
      key "identifier";
      description
        "Content of a subscription.";
      leaf identifier {
        type subscription-id;
        description
          "Identifier to use for this subscription.";
      }
      uses subscription-policy-non-modifiable;
      uses subscription-policy-modifiable;
      uses receiver-info {
        if-feature "configured-subscriptions";
      }
      uses notification-origin-info {
        if-feature "configured-subscriptions";
      }
    }
  }
  container subscriptions {
    config false;
    description
```

```
      "Contains the list of currently active subscriptions, i.e.
       subscriptions that are currently in effect, used for subscription
       management and monitoring purposes. This includes subscriptions
       that have been setup via RPC primitives as well as subscriptions
       that have been established via configuration.";
    list subscription {
      key "identifier";
      config false;
      description
        "Content of a subscription.
         Subscriptions can be created using a control channel or RPC, or
         be established through configuration.";
      leaf identifier {
        type subscription-id;
        description
          "Identifier of this subscription.";
      }
      leaf configured-subscription {
       if-feature "configured-subscriptions";
       type empty;
       description
         "The presence of this leaf indicates that the subscription
          originated from configuration, not through a control channel
          or RPC.";
      }
      uses subscription-policy;
      uses notification-origin-info {
        if-feature "configured-subscriptions";
      }
      uses receiver-info {
        augment receivers/receiver {
          description
            "include operational data on configured receivers.";
          leaf pushed-notifications {
            type yang:counter64;
            description
              "Operational data which provides the number of update
               notifications pushed to a receiver.";
            }
          leaf excluded-notifications {
            type yang:counter64;
            description
              "Operational data which provides the number of non-
               datastore update notifications explicitly removed via
               filtering so that they are not sent to a receiver.";
            }
        }
      }
      leaf subscription-status {
        type subscription-status;
        description
          "The status of the subscription.";
      }
    }
  }
}
<CODE ENDS>
```

10. Considerations

10.1. Implementation Considerations

For a deployment including both configured and dynamic subscriptions, split subscription identifiers into static and dynamic halves. That way there should not be collisions if the configured subscriptions attempt to set a subscription-id which might have already been dynamically allocated.

The <notification> elements are never sent before the transport layer, including capabilities exchange, has been established.

10.2. Security Considerations

A secure transport is highly recommended and the publisher must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of content involved. When a <get> is received that refers to the content defined in this memo, recievers should only be able to view the content for which they have sufficient privileges. <establish-subscription> operations can be considered like deferred <get>, and the content that different users can access may vary. This different access is reflected in the <notificationt> to which different users are able to subscribe.

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 publisher MUST NOT include any content in a notification that the user is not authorized to view.

If a malicious or buggy subscriber sends a number of <establish-subscription> requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions can be terminated by terminating the transport session. The publisher can also suspend or terminate subscriptions with per-subscription granularity.

A subscription could be configured on another receiver’s behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver, which doesn’t even support subscriptions. Subscribers that do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, 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.

Note that streams can define additional authorization requirements. For instance, in [I-D.ietf-netconf-yang-push], each of the elements in its data plane notifications must also go through access control.

It is recommended that the NACM "very-secure" tag is placed on the <kill-subscription> RPC so that only administrators can access.

11. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Tim Jenkins, Balazs Lengyel, Shaon Chisholm, Hector Trevino, Susan Hares, Kent Watsen, Michael Scharf, and Guangying Zheng.

12. References

12.1. 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>.

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

[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>.

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

12.2. Informative References

[I-D.ietf-netconf-netconf-event-notif]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF support for event notifications", August 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>.

[I-D.ietf-netconf-restconf-notif]
Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", August 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf-notif/>.

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

[I-D.voit-notifications2]
Voit, Eric., Clemm, Alexander., Bierman, A., and T. Jenkins, "YANG Notification Headers and Bundles", February 2017, <https://datatracker.ietf.org/draft-voit-netmod-yang-notifications2/>.

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

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

Appendix A. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

Issue #9: validate that Subscription ID will only be relevant locally to a single receiver

Issue #6: Data plane notifications and layered headers

How to allow for seamless integration with non-standard encodings and transports (like GPB/GRPC). Specify requirements encoding and transport must meet, provide examples.

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v01 5277bis - v00 subscribed notifications

- Kill subscription RPC added.
- Renamed from 5277bis to Subscribed Notifications.
- Changed the notification capabilities version from 1.1 to 2.0 as this is not RFC-5277 compatible.
- Extracted create-subscription and other elements of RFC5277.
- Error conditions added, and made specific in return codes.
- Simplified yang model structure for removal of ’basic’ grouping.
- Added a grouping for items which cannot be statically configured.
- Streams extracted if favor of more information in the filters section.
- Operational counters per receiver.
- Subscription-id and filter-id renamed to identifier
- Section for replay added. Replay-start and stop-time updated. Replay now cannot be configured.
- Control plane notification renamed to subscription state notification
- Souce address: Source-vrf changed to string, default address option added
- In yang model: ’info’ changed to ’policy’
- Scattered text clarifications

v00 - v01 of 5277bis

- o YANG Model changes. New groupings for subscription info to allow restriction of what is changable via RPC. Removed notifications for adding and removing receivers of configured subscriptions.

- o Expanded/renamed defintions from event server to publisher, and client to subscriber as applicable. Updated the definitions to include and expand on RFC 5277.

- o Removal of redundant with other drafts

- o Many other clean-ups of wording and terminology

Authors’ Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com