NETCONF
Internet-Draft
Intended status: Standards Track
Expires: March 23, 2018

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
VMWare
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
September 19, 2017

Custom Subscription to Event Notifications
draft-ietf-netconf-subscribed-notifications-04

Abstract

This document defines capabilities and operations for the customized establishment of subscriptions upon a publisher's event streams. Also defined are delivery mechanisms for instances of the resulting events. Effectively this allows a subscriber to request and receive a continuous, custom influx of publisher generated information.

Status of This Memo

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

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

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

This Internet-Draft will expire on March 23, 2018.

Table of Contents

1. Introduction

This document defines capabilities and operations for the customized establishment of subscriptions upon system generated event streams. Effectively this enables a "Subscribe then Publish" capability where the customized information needs of each target receiver are understood by the publisher before events are marshalled and pushed. The receiver then gets a continuous, custom influx of publisher generated events.

While the functionality defined in this document is transport-agnostic, subscription control plane operations bindings exist for both NETCONF [RFC6241] and RESTCONF [RFC8040]. In addition, bindings for the pushed event instances have been defined for protocols such as NETCONF and HTTP2 [RFC7540]. For specifics on these bindings see [I-D.ietf-netconf-event-notif]) and [I-D.ietf-netconf-restconf-notif].

1.1. Motivation

There are various [RFC5277] limitations, many of which have been exposed in [RFC7923] which needed to be solved. Key capabilities supported by this document include:

- multiple subscriptions on a single transport session
- support for dynamic and statically configured subscriptions
- modification of an existing subscription
- operational counters and instrumentation
- negotiation of subscription parameters
- promise theory based interaction model
- state change notifications (e.g., publisher driven suspension, parameter modification)
- independence from transport protocol

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.)

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

NACM: NETCONF Access Control Model.

Notification message: 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.

Publisher: An entity responsible for streaming notification messages per the terms of a Subscription.

Receiver: A target to which a publisher pushes subscribed content. For dynamic subscriptions, the receiver and subscriber are the same entity.

Stream (also referred to as "event stream"): A continuous ordered set of events aggregated under some context.

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

Subscription: A contract with a publisher, stipulating which information one or more receivers wish to have pushed from the publisher without the need for further solicitation.

1.3. Solution Overview

This document describes a transport protocol-agnostic mechanism for subscribing to and receiving content from a stream instantiated within an event publisher. This mechanism is through the use of a subscription.

Two types of subscriptions are supported:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via RPC. If the publisher wants to serve this request, it accepts it, and then starts pushing notification messages. 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 allow the management of subscriptions via a configuration interface so that a publisher can send notification messages to configured receiver(s). Support for this capability is optional.

Additional characteristics differentiating configured from dynamic subscriptions include:

- The lifetime of a dynamic subscription is bounded by the transport session used to establish it. For connection-oriented stateful transport like NETCONF, the loss of the transport session will result in the immediate termination of associated dynamic subscriptions. For connectionless or stateless transports like HTTP, it is the lack of receipt acknowledgement of a sequential set of notification messages and/or keep-alives which will terminate dynamic subscriptions. The lifetime of a configured subscription is driven by relevant configuration being present on the running configuration. This implies configured subscriptions persist across reboots, and persist even when transport is unavailable.

- Configured subscriptions can be modified by any configuration client with write permission on the configuration of the subscription. Dynamic subscriptions can only be modified via an RPC request made upon the original subscribing transport session.

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 (if needed, an operator may use a kill RPC however).

The publisher MAY decide to terminate a dynamic subscription at any time. Similarly, it MAY decide to temporarily suspend the sending of notification messages 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 named entity on a publisher which exposes a continuously updating set of events. Each event stream is available for subscription. 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.

There are two standardized event streams within this document: NETCONF and SYSLOG. The NETCONF event stream contains all NETCONF XML event information supported by the publisher, except for where it has been explicitly indicated that this the event MUST be excluded from the NETCONF stream. The SYSLOG event stream mirrors the discrete set entries which are concurrently being placed into a device's local Syslog. Beyond these two, additional streams can be added via model augmentation.

As events are raised by a system, they may be assigned to one or more streams. The event is distributed to receivers where: (1) a subscription includes the identified stream, and (2) subscription filtering allows the event to traverse.

If access control permissions are in use to secure publisher content, then for notifications to be sent to a receiver, that receiver MUST be allowed access to all the events on the stream. If permissions change during the lifecycle of a subscription, then events MUST be sent or restricted accordingly. This can be done by re-establishing a subscription with the updated permissions, or by seamlessly updating the permissions of an existing subscription.

2.2. Event Filters

A publisher implementation MUST support the ability to perform filtering of events within a stream. Two filtering syntaxes supported are [XPATH] and subtree [RFC6241]. The subsets of these filtering syntaxes supported are left to each implementation. Events which evaluate to "true", or return a non-null selection as a result of the evaluation by the event filter MUST traverse the filter in

their entirety. A subset of information is never stripped from within the event.

If no event filter is provided, all events on a stream are to be sent.

2.3. Subscription State Model at the Publisher

Below is the state machine of a subscription for the publisher for a dynamic subscription. It is important to note that such 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.

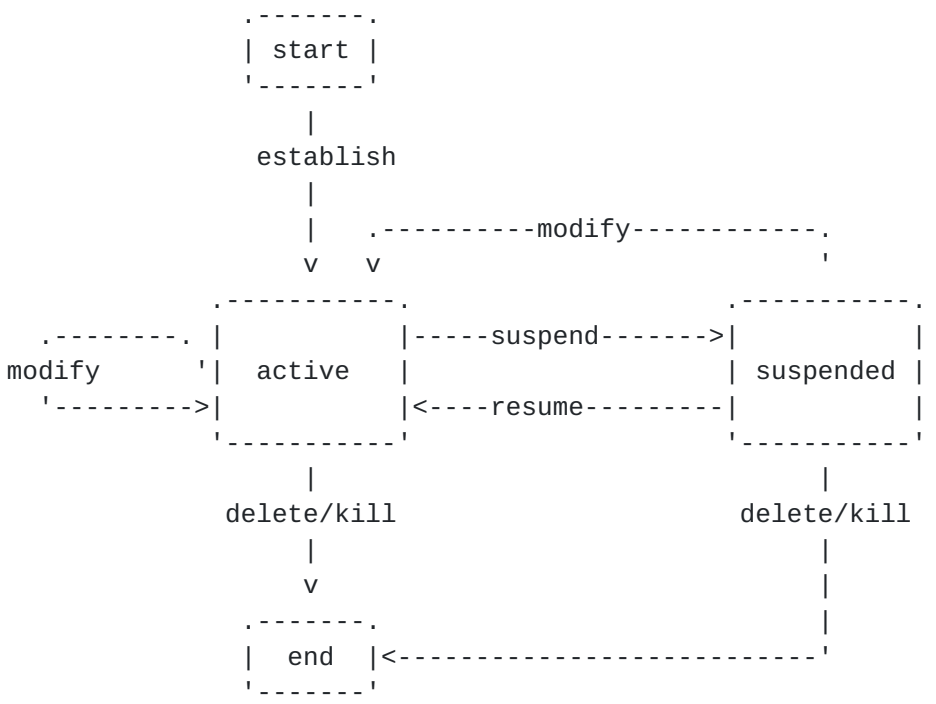

Figure 1: Subscription states at publisher

Of interest in this state machine are the following:

- Successful establish or modify RPCs put the subscription into an active state.
- Failed modify RPCs will leave the subscription in its previous state, with no visible change to any streaming updates.
- A delete or kill RPC will end the subscription.

- Suspend and resume state changes are driven by internal process and prioritization. There are no external controls over suspend and resume.

An equivalent state machine exists for configured subscriptions. However the transition between states is via configuration operations rather than via RPC.

3. Data Model Trees

```
module: ietf-subscribed-notifications
    +--ro streams
    |  +--ro stream* [stream]
    |     +--ro stream                     stream
    |     +--ro description                string
    |     +--ro replay-support?            empty
    |     +--ro replay-log-creation-time?  yang:date-and-time
    |     +--ro replay-log-aged-time?      yang:date-and-time
    +--rw filters
    |  +--rw filter* [identifier]
    |     +--rw identifier              filter-id
    |     +--rw (filter-type)?
    |        +--:(event-filter)
    |           +--rw event-filter-type       event-filter-type
    |           +--rw event-filter-contents
    +--rw subscription-config {configured-subscriptions}?
    |  +--rw subscription* [identifier]
    |     +--rw identifier              subscription-id
    |     +--rw encoding?               encoding
    |     +--rw (target)
    |     |  +--:(stream)
    |     |     +--rw (event-filter)?
    |     |     |  +--:(by-reference)
    |     |     |  |  +--rw filter-ref              filter-ref
    |     |     |  +--:(within-subscription)
    |     |     |     +--rw event-filter-type       event-filter-type
    |     |     |     +--rw event-filter-contents
    |     |     +--rw stream                  stream
    |     |     +--rw replay-start-time?   yang:date-and-time {replay}?
    |     +--rw stop-time?              yang:date-and-time
    |     +--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 encoding?                  encoding
         +--ro (target)
         |  +--:(stream)
         |     +--ro (event-filter)?
         |     |  +--:(by-reference)
         |     |  |  +--ro filter-ref                filter-ref
         |     |  +--:(within-subscription)
         |     |     +--ro event-filter-type          event-filter-type
         |     |     +--ro event-filter-contents
         |     +--ro stream                      stream
         |     +--ro replay-start-time?    yang:date-and-time {replay}?
         +--ro stop-time?                 yang:date-and-time
         +--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 port]
               +--ro address                   inet:host
               +--ro port                      inet:port-number
               +--ro protocol?                 transport-protocol
               +--ro pushed-notifications?     yang:counter64
               +--ro excluded-notifications?   yang:counter64
               +--ro status                    subscription-status

  rpcs:
    +---x establish-subscription
    |  +---w input
    |  |  +---w encoding?               encoding
    |  |  +---w (target)
    |  |  |  +--:(stream)
    |  |  |     +---w (event-filter)?
    |  |  |     |  +--:(by-reference)
    |  |  |     |  |  +---w filter-ref              filter-ref
    |  |  |     |  +--:(within-subscription)
    |  |  |     |     +---w event-filter-type       event-filter-type
    |  |  |     |     +---w event-filter-contents
    |  |  |     +---w stream                  stream
```

```
   |  |  |     +---w replay-start-time?    yang:date-and-time {replay}?
   |  |  +---w stop-time?               yang:date-and-time
   |  +--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 (target)
   |  |  |  +--:(stream)
   |  |  |     +---w (event-filter)?
   |  |  |        +--:(by-reference)
   |  |  |        |  +---w filter-ref              filter-ref
   |  |  |        +--:(within-subscription)
   |  |  |           +---w event-filter-type       event-filter-type
   |  |  |           +---w event-filter-contents
   |  |  +---w stop-time?              yang:date-and-time
   |  +--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-completed
   |  +--ro identifier    subscription-id
   +---n subscription-completed
   |  +--ro identifier    subscription-id
   +---n subscription-started
   |  +--ro identifier              subscription-id
   |  +--ro encoding?               encoding
   |  +--ro (target)
   |  |  +--:(stream)
```

```
   |  |     +--ro (event-filter)?
   |  |     |  +--:(by-reference)
   |  |     |  |  +--ro filter-ref              filter-ref
   |  |     |  +--:(within-subscription)
   |  |     |     +--ro event-filter-type       event-filter-type
   |  |     |     +--ro event-filter-contents
   |  |     +--ro stream                  stream
   |  |     +--ro replay-start-time?      yang:date-and-time {replay}?
   |  +--ro stop-time?              yang:date-and-time
   +---n subscription-resumed
   |  +--ro identifier    subscription-id
   +---n subscription-modified
   |  +--ro identifier              subscription-id
   |  +--ro encoding?               encoding
   |  +--ro (target)
   |  |  +--:(stream)
   |  |     +--ro (event-filter)?
   |  |     |  +--:(by-reference)
   |  |     |  |  +--ro filter-ref              filter-ref
   |  |     |  +--:(within-subscription)
   |  |     |     +--ro event-filter-type       event-filter-type
   |  |     |     +--ro event-filter-contents
   |  |     +--ro stream                  stream
   |  |     +--ro replay-start-time?      yang:date-and-time {replay}?
   |  +--ro stop-time?              yang:date-and-time
   +---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 top-level decompositions of data model are as follows:

- "Streams" contains a list of event streams that are supported by the publisher and against which subscription is allowed.

- "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 a 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 stream which identifies the domain of events against which the subscription is applied.

- A filter which may reduce the set of events pushed.

- 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 a non-existent stream is referenced.

4.1.1. Replay Subscription

Replay provides the ability to establish an event subscription which is also capable of passing along recently generated events. In other words, as the subscription initializes itself, it sends any previously generated notifications for the target event stream which meet the filter and timeframe criteria. These historical events would then be followed by events generated after the subscription has been established. All events will be delivered in the order generated. Replay is only viable for dynamic subscriptions. Replay is an optional feature. Replay is dependent on a notification stream supporting some form of notification logging, although it puts no restrictions on the size or form of the log, or where it resides within the device.

The presence of a replay-start-time within an <establish-subscription> RPC is the way a replay may be been requested. The provided 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. An end time MAY be specified using the optional stop-time parameter, which MAY also be earlier than the current time. If no stop-time is present, notifications will continue to be sent until the subscription is terminated.

Not all streams will support replay. Those that do MUST include the <replay-supported> flag. In addition, a notification stream that does support replay is not expected to have an unlimited supply of saved notifications available to accommodate any given replay request. Subscribers MAY do a get on <replay-log-creation-time> and <replay-log-aged-time> to assess the availability of notifications for replay. The actual number of stored notifications available for retrieval at any given time is a publisher specific matter. Control parameters for this aspect of the feature are outside the scope of this document.

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 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.

4.4. Killing a Subscription

The <kill-subscription> operation permits an operator to end any dynamic subscription. A publisher MUST terminate any dynamic subscription identified within a properly authenticated RPC request.

Configured subscriptions cannot be killed 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 plus VRF out of which subscription updates should be pushed from the publisher. Where these are not explicitly included, or where just the VRF is provided, push updates MUST egress the publisher's default interface towards a receiver.

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

Immediately after a subscription is successfully established, the publisher sends to identified receivers a state change notification stating the subscription has been established (i.e., subscription-started).

It is quite possible that upon configuration, reboot, or even steady-state operations, a transport session may not be currently available to the receiver. In this case, when there is something to transport for an active subscription, transport protocol specific call-home operations will be used to establish the connection.

As an example at subscription establishment using <edit-config> over NETCONF, a client might 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:yang:"+
               "ietf-subscribed-notifications:1.0">
            <subscription>
                <subscription-id>1922</subscription-id>
                <stream>NETCONF</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 state change 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 state change 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 state change

notification stating the subscription has been terminated (i.e., subscription-terminated).

6. Deleting a Configured Subscription

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

Immediately after a subscription is successfully deleted, the publisher sends to the existing receivers a state change notification stating the subscription has been terminated (i.e., subscription-terminated).

7. Asynchronous Subscribed Content Delivery

Once a subscription has been set up, the publisher streams asynchronously push subscribed content via notification messages per the terms of the subscription. For dynamic subscriptions set up via RPC operations, notification messages are sent over the session used to establish the subscription. For configured subscriptions, notification messages are sent over the specified connections.

A notification message is sent to a receiver when something of interest occurs which is able to traverse all specified filtering and access control criteria.

This notification message MAY be encoded as one-way notification element of [RFC5277], Section 4. The following example within [RFC7950] section 7.16.3 is an example of a compliant message:

```
<notification
       xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <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: subscribed content notification

This [RFC5277] section 4 has the drawback of not including useful header information such as subscription-ID. When using this mechanism, it is left up to implementations to determine which events belong to which subscription.

This drawback, along with other useful common headers and the ability to bundle multiple event notifications together is being explored within [I-D.notification-messages].

8. Subscription State Notifications

In addition to subscribed content, a publisher will 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 events has been completed, that a subscription is done 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.

8.1. 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.

8.2. 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.

8.3. 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. Northbound systems MAY 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 subscription, then there MUST be a notification that the subscription has been ended.

8.4. 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 subscribed content 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).

8.5. subscription-resumed

This notification indicates that a previously suspended subscription has been resumed. Subscribed content generated after the generation of this state change notification will be sent. These notifications go to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

8.6. subscription-completed

This notification is sent to indicate that a subscription, which includes a stop time, has successfully finished passing events upon the reaching of that stop time.

8.7. replay-completed

This notification indicates that all of the events prior to the current time have been sent. This includes new events 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-completed notification has been sent events will be sent in sequence as they arise naturally within the system.

9. Administrative Functions

9.1. Subscription Monitoring

Container "subscriptions" in the YANG module below contains the state of all known subscriptions. 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, receiver counter information, the status of the subscription, as well as the various subscription parameters that are in effect. The subscription status indicates the subscription's state with each receiver (e.g., is currently active or suspended). 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.

9.2. Capability Advertisement

Capabilities are advertised in messages sent by each peer during session establishment [RFC6241]. Publishers supporting the RPCs and Notifications in this document MUST advertise the capability "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications:1.0". In addition support for optional features: json, configured-subscriptions, and replay MAY also be advertised.

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 any new subscription capabilities defined in this document.

If a publisher supports this specification but not [RFC5277], the publisher MUST support the one-way notification element of [RFC5277] Section 4 or [I-D.notification-messages].

9.3. 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.

10. Data Model

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

     Editor:   Alexander Clemm
               <mailto:ludwig@clemm.org>

     Editor:   Eric Voit
               <mailto:evoit@cisco.com>

     Editor:   Alberto Gonzalez Prieto
               <mailto:agonzalezpri@vmware.com>

     Editor:   Einar Nilsen-Nygaard
               <mailto:einarnn@cisco.com>

     Editor:   Ambika Prasad Tripathy
               <mailto:ambtripa@cisco.com>";


  description
    "Contains a conceptual YANG specification for subscribing to events
    and receiving matching content within notification messages.";

  revision 2017-09-15 {
```

```
    description
      "Filtering and stream structures updated, replay a feature.";
    reference
      "draft-ietf-netconf-subscribed-notifications-04";
  }

  /*
   * 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.";
  }

  feature replay {
    description
      "This feature indicates that historical event replay is
      supported.  With replay, it is possible for past events to be
      will be streamed in chronological order.";
  }

  /*
   * EXTENSIONS
   */

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

  /*
   * IDENTITIES
   */

  /* Identities for streams */
  identity stream {
```

```
    description
      "Base identity to represent a generic stream of event
       notifications exposed for subscription by a system.";
  }

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

  identity SYSLOG {
    base stream;
    description
      "A stream of events mirroring the discrete set entries
      concurrently being placed into a device's local Syslog.";
  }

  identity custom-stream {
    base stream;
    description
      "A supported stream not defined via an identity in this model";
  }

  /* Identities for event filters */

  identity event-filter {
    description
      "Evaluation criteria used as a pass/fail test against events.
      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
      for its value to be checked against, the event will be filtered
      out.  For example, if one were to check for 'severity=critical' in
      an event where this object does not exist, then the event would
      not traverse.";
  }

  identity subtree-event-filter {
    base event-filter;
    description
      "An RFC-6241 based filter which attempts to select nodes within an
      event.  After evaluation, the return of a non-empty node set means
      that the filter is successfully traversed.";
    reference "RFC-6241, #5.1";
  }
```

```
  identity xpath-event-filter {
    base event-filter;
    description
      "A filter applied to an event which follows the syntax specified
      in yang:xpath1.0. Success is indicated by either a positive
      boolean result, or a non-null node selection.";
    reference "XPATH: http://www.w3.org    R/1999/REC-xpath-19991116";
  }

  /* Identities for subscription results */
  identity subscription-result {
    description
      "Base identity for RPC responses and State Change Notifications
      providing information on the 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
      "Problem with subscription.  Base identity for error return
      codes for RPCs and State Change Notifications.";
  }

  /* Identities for subscription stream status */
  identity subscription-status {
    description
      "Base identity for the status of a subscription.  This status
      indicates whether a subscriber is actively generating
      notifications intended for a particular receiver.";
  }

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

  identity inactive {
    base subscription-status;
    description
      "Status is inactive, for example after the stop time, but not
```

```
      yet deleted from the configuration.";
  }

  identity suspended {
    base subscription-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-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 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-type-unsupported {
    base error;
    description
     "Publisher does not support filters constructed using this
     filtering technology syntax.";
  }

  identity filter-unsupported {
    base error;
    description
     "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.";
```

```
  }

  identity error-insufficient-resources {
    base error;
    description
      "The server has insufficient resources to support the
       subscription as requested by an RPC.";
  }

  identity unsupportable-volume {
    base error;
    description
      "The publisher cannot support the volume of information intended
      to be sent for an existing subscription.";
  }

  identity error-no-such-option {
    base error;
    description
      "A requested parameter setting is not supported.";
  }

  /* 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.";
  }

  identity http2 {
    base transport;
    description
      "HTTP2 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 event-filter-type {
    type identityref {
      base event-filter;
    }
    description
      "Specifies a known type of event filter.";
  }

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

  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.";

       leaf event-filter-type {
         type event-filter-type;
         mandatory true;
         description
           "A filter needs to be a known and understood syntax if it is
           to be interpretable by a device.";
       }
       anyxml event-filter-contents {
         mandatory true;
         description
           "Event stream evaluation criteria encoded in a syntax of a
           supported type of filter.  If the filter is applied
           against an event stream and there is a non-empty or
           positive result, the event is passed along.";
       }
  }

  grouping subscription-policy-modifiable {
    description
      "This grouping describes all objects which may be changed
      in a subscription via an RPC.";
    choice target {
      mandatory true;
```

```
      description
        "A filter must be applied against some source of information.
        This identifies the target for the filter.";
      case stream {
        choice event-filter {
          description
            "A filter can be applied to a subscription.  And that filter
            will come either referenced from a global list, or be
            provided within the subscription itself.";
          case by-reference {
            description
              "Apply a filter that has been configured separately.";
            leaf filter-ref {
              type filter-ref;
              mandatory true;
              description
                "References an existing filter which is to be applied to
                the subscription.";
            }
          }
          case within-subscription {
            description
              "Local definition allows a filter to have the same
              lifecycle as the subscription.";
            uses base-filter;
          }
        }
      }
    }
    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.";
    }
  }

  grouping subscription-policy {
    description
      "This grouping describes information concerning a subscription.";
    leaf encoding {
      type encoding;
      default "encode-xml";
      description
```

```
        "The type of encoding for the subscribed data. Default is XML";
    }
    uses subscription-policy-modifiable {
      augment target/stream {
        description
          "Adds additional objects which must be set just by RPC.";
        leaf stream {
          type stream;
          mandatory true;
          description
            "Indicates a stream of events against which to apply
            an event filter.";
        }
        leaf replay-start-time {
          if-feature "replay";
          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 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
            is not complete, and might not be implemented on a box.";
        }
        leaf source-address {
          type inet:ip-address-no-zone;
          description
            "The source address for the notifications.  If a source VRF
            exists, but this object doesn't, a default address for the
            VRF can be used.";
        }
      }
    }
  }

  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 receiver.";
    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;
          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;
          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 hints {
    description
      "Objects 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 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 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 reply MAY include suggested
       parameter settings that would have a higher 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 has been deleted, 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 has been killed, or if a
           problem was encountered.";
      }
    }
  }

  /*
   * NOTIFICATIONS
   */

  notification replay-completed {
    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 subscription-completed {
    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 gracefully completed 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.  For completeness, this
       notification includes both modified and non-modified aspects of
       a 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 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 hints;
  }

  /*
   * DATA NODES
   */

  container streams {
    config false;
    description
      "This container contains information on the built-in streams
      provided by the publisher.";
    list stream {
      key "stream";
      description
        "Identifies the built-in streams that are supported by the
         publisher.";
      leaf stream {
        type stream;
        description
          "A handle for a sequential set of events, each of which
           is characterized by its own domain and semantics.
           In case configurable custom streams are supported,
           as indicated by the custom-stream identity, the configuration
           of those custom streams is provided separately.";
      }
      leaf description {
        type string;
        mandatory true;
```

```
        description
          "A description of the event stream, including such information
           as the type of events that are sent over this stream.";
      }
      leaf replay-support {
        type empty;
        description
          "Indicates that event replay is available on this stream.";
      }
      leaf replay-log-creation-time {
        type yang:date-and-time;
        description
          "The timestamp of the creation of the log used to support the
          replay function on this stream. Note that this might be
          earlier then the earliest available notification in the log.
          This object is updated if the log resets for some reason.
          This object MUST be present if replay is supported.";
      }
      leaf replay-log-aged-time {
        type yang:date-and-time;
        description
          "The timestamp of the last notification aged out of the log.
          This object MUST be present if replay is supported and any
          notifications have been aged out of the log.";
      }
    }
  }

  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.";
      }
      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 event-filter {
          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;
      uses receiver-info;
      uses notification-origin-info;
    }
  }
  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";
      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 a subscription; unique within a publisher";
      }
      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 for 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 events
               from a stream explicitly removed via filtering so that
               they are not sent to a receiver.";
            }
          leaf status {
            type subscription-status;
            mandatory true;
            description
              "The status of the subscription.";
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

11. Considerations

11.1. Implementation Considerations

For a deployment including both configured and dynamic subscriptions, split subscription identifiers into static and dynamic halves. That way it is unlikely there will be collisions if the configured subscriptions attempt to set a subscription-id which might have

already been dynamically allocated. The lower half SHOULD be used for configured subscriptions and upper half for dynamic.

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

It is left to an implementation to determine when to transition between active and suspended subscription states. However if a subscription is unable to marshal all intended updates into a transmittable message in multiple successive intervals, the subscription SHOULD be suspended with the reason "unsupportable-volume".

11.2. Security Considerations

For dynamic subscriptions the publisher MUST authenticate and authorize all RPC requests.

Subscriptions could overload a publisher's CPU. For this reason, the publisher MUST have the ability to decline a dynamic subscription request, and provide the appropriate RPC error response to a subscriber should the proposed subscription overly deplete the publisher's resources.

A publisher needs to be able to suspend an existing dynamic or configured subscription based on capacity constraints. When this occurs, the subscription status MUST be updated accordingly and the receivers notified with subscription state notifications.

If a malicious or buggy subscriber sends an unexpectedly large number of RPCs, the result might be an exessive use of system resources. In such a situation, subscription interactions MAY be terminated by terminating the transport session.

For both configured and dynamic subscriptions the publisher MUST authenticate and authorize a receiver via some transport level mechanism before sending any updates.

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.

A publisher MUST NOT include any content in a notification for which the user has not been authorized.

With configured subscriptions, one or more publishers could be used to overwhelm a receiver. No push updates SHOULD be sent to any

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.

The NETCONF Authorization Control Model [RFC6536bis] SHOULD be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user permissions to receive events from specific streams.

Where NACM is available, the NACM "very-secure" tag MUST be placed on the <kill-subscription> RPC so that only administrators have access to use this.

One subscription id can be used for two or more receivers of the same configured subscription. But due to the possibility of different access control permissions per receiver, it SHOULD NOT be assumed that each receiver is getting identical updates.

12. Acknowledgments

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

13. References

13.1. Normative References

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

[RFC6536bis] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-01 (work in progress), September 2017.

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

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

13.2. Informative References

[I-D.ietf-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., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", June 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

[I-D.notification-messages]
Voit, Eric., Clemm, Alexander., Bierman, A., and T. Jenkins, "YANG Notification Headers and Bundles", September 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-notification-messages>.

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

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

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

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

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

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

v03 - v04

- Moved back to the use of RFC5277 one-way notifications and encodings.

v03 - v04

- Replay updated

v02 - v03

- RPCs and Notification support is identified by the Notification 2.0 capability.
- Updates to filtering identities and text
- New error type for unsupportable volume of updates
- Text tweaks.

v01 - v02

- Subscription status moved under receiver.

v00 - v01

- Security considerations updated
- Intro rewrite, as well as scattered text changes
- Added Appendix A, to help match this to related drafts in progress
- Updated filtering definitions, and filter types in yang file, and moved to identities for filter types
- Added Syslog as a stream
- HTTP2 moved in from YANG-Push as a transport option

- Replay made an optional feature for events. Won't apply to datastores
- Enabled notification timestamp to have different formats.
- Two error codes added.

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.
- 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.
- Operational counters per receiver.
- Subscription-id and filter-id renamed to identifier
- Section for replay added. Replay now cannot be configured.
- Control plane notification renamed to subscription state notification
- Source address: Source-vrf changed to string, default address option added
- In yang model: 'info' changed to 'policy'
- Scattered text clarifications

v00 - v01 of 5277bis

- YANG Model changes. New groupings for subscription info to allow restriction of what is changeable via RPC. Removed notifications for adding and removing receivers of configured subscriptions.
- Expanded/renamed definitions from event server to publisher, and client to subscriber as applicable. Updated the definitions to include and expand on RFC 5277.

- Removal of redundancy with other drafts
- 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
VMWare

Email: agonzalezpri@vmware.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com