NETCONF
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2016

A. Gonzalez Prieto
A. Clemm
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
S. Chisholm
Ciena
H. Trevino
Cisco Systems
June 15, 2016

Subscribing to YANG-Defined Event Notifications
draft-gonzalez-netconf-5277bis-02

Abstract

This document defines capabilities and operations for providing asynchronous message notification delivery for notifications defined using YANG. 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 along with their mapping onto NETCONF transport (to be specified in a separate document, but still included in the current document version) are intended to obsolete RFC 5277.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 17, 2016.

Table of Contents

1. Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service for the NETCONF protocol . This is an optional capability built on top of the base NETCONF definition. This document defines capabilities and operations for providing asynchronous message notification delivery for notifications defined using YANG, including capabilities and operations 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] and Restconf [I-D.ietf-netconf-restconf]. The capabilities and operations defined in this document along with their mapping onto NETCONF transport (to be specified in a separate document, but still included in the current document version) are intended to obsolete RFC 5277.

Editor's note: The current version of this document specifies both capabilities and operations for providing asynchronous notification delivery, as well as mapping of those capabilities and operations onto NETCONF. The transport mapping to NETCONF will be moved into a separate document and will be removed in future revisions of this document.

1.1. Motivation

The motivation for this work is to enable the sending of asynchronous notification messages that are consistent with the data model (content) and security model used within a NETCONF implementation.

[RFC5277] defines a notification mechanism for NETCONF. However, there are various limitations:

- Each subscription requires a separate NETCONF connection, which is wasteful.
- The only mechanism to terminate a subscription is terminating the underlying NETCONF connection.
- No ability to modify subscriptions once they have been created.
- No ability to notify the receiver of a subscription if the server is dropping events.
- No mechanism to monitor subscriptions.
- No alternative mechanism to create subscriptions via RPCs. Thus the lifetime of the subscription is limited by that of the underlaying NETCONF session.
- Predates YANG and defines RPCs, notifications, and data nodes outside of the YANG framework.

The scope of the work aims at meeting the following operational needs:

- Ability to dynamically or statically subscribe to event notifications available on a NETCONF agent.
- Ability to negotiate acceptable dynamic subscription parameters.
- Ability to support multiple subscriptions over a single NETCONF session.
- Ability to filter the subset of notifications to be pushed with stream-specific semantics.
- Ability for the notification payload to be interpreted independently of the NETCONF transport protocol. (In other words, the encoded notification fully describes itself.)
- Mechanism to communicate the notifications.
- Ability to replay locally logged notifications.
- Backwards compatible with RFC 5277 implementations.

- Define in YANG, the RPCs, notifications, and data nodes in RFC 5277.

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].

Event: Something that happens 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 message sent by a server to a receiver indicating that an event (of interest to the subscriber) has occurred. Events can trigger notifications if an interested party has subscribed to the stream(s) it belongs to.

Stream (also referred to as "event stream"): A continuous flow of event, status, state, or other information.

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

Receiver: A target to which a NETCONF server pushes event notifications. In many deployments, the receiver and subscriber will be the same entity.

Subscription: A contract between a subscriber and a NETCONF server, stipulating which information the receiver wishes to have pushed from the server without the need for further solicitation.

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.

Dynamic subscription: A subscription agreed between subscriber and NETCONF server via create, establish, modify, and delete RPC control plane signaling messages.

Configured subscription: A subscription installed via a configuration interface.

Operation: In this document, this term refers to NETCONF protocol operations [RFC6241] defined in support of NETCONF notifications.

NACM: NETCONF Access Control Model.

RPC: Remote Procedure Call.

1.3. Solution Overview

This document describes mechanisms for subscribing and receiving event notifications from a NETCONF server. This document enhances the capabilities of RFC 5277 while maintaining backwards capability with existing implementations. It is intended that a final version of this document might obsolete RFC 5277.

The enhancements over [RFC5277] include the ability to terminate subscriptions without terminating the client session, to modify existing subscriptions, and to have multiple subscriptions on a NETCONF session.

These enhancements do not affect [RFC5277] clients that do not support these particular subscription requirements.

The solution supports subscribing to event notifications using two mechanisms.

1. Dynamic subscriptions, where a NETCONF client initiates a subscription negotiation with a NETCONF server. Here a client initiates a negotiation by issuing a subscription request. If the agent wants to serve this request, it will accept it, and then start pushing event notifications as negotiated. If the agent does not wish to serve it as requested, it may respond with subscription parameters, which it would have accepted.

2. Configured subscriptions, which is an optional mechanism that enables managing subscriptions via a configuration interface so that a NETCONF agent sends event notifications to given receiver(s).

Some key characteristics of configured and dynamic subscriptions include:

- 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.

- 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. This also means configured subscriptions do not support negotiation.

- 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.

- A NETCONF agent can support multiple dynamic subscriptions simultaneously in the context of a single NETCONF session. (This requires supporting interleaving.) The termination of any of those subscriptions does not imply the termination of NETCONF transport session.

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

The NETCONF agent may decide to terminate a dynamic subscription at any time. Similarly the NETCONF agent 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 agent running out of resources to serve the subscription, or by internal errors on the server.

2. Solution

2.1. Event Streams

An event stream is a set of events available for subscription from a server. 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.

The following is a high-level description of the flow of a notification. Note that it does not mandate and/or preclude an implementation. As events are raised, they are assigned to streams. An event may be assigned to multiple streams. The event is distributed to subscribers and receivers based on the current subscriptions and access control. Access control is needed because if any receiver of that subscription does not have permission to receive an event, then it never makes it into a notification, and processing of the event is completed for that subscription.

2.2. Event Stream Discovery

A server maintains a list of available event streams as operational data. A client can retrieve this list like any other YANG-defined data, for example using the <get> operation when using NETCONF.

2.3. Default Event Stream

A NETCONF server implementation supporting the notification capability MUST support the "NETCONF" notification event stream. This stream contains all NETCONF XML event notifications supported by the NETCONF server, except for those belonging only to streams that explicitly indicate that they must be excluded from the NETCONF stream. The exact string "NETCONF" is used during the advertisement of stream support during the <get> operation on <streams> and during the <create-subscription> and <establish-subscription> operations.

2.4. Filters

A NETCONF Server implementation SHOULD support the ability to perform filtering of notification records per RFC 5277.

2.5. 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.

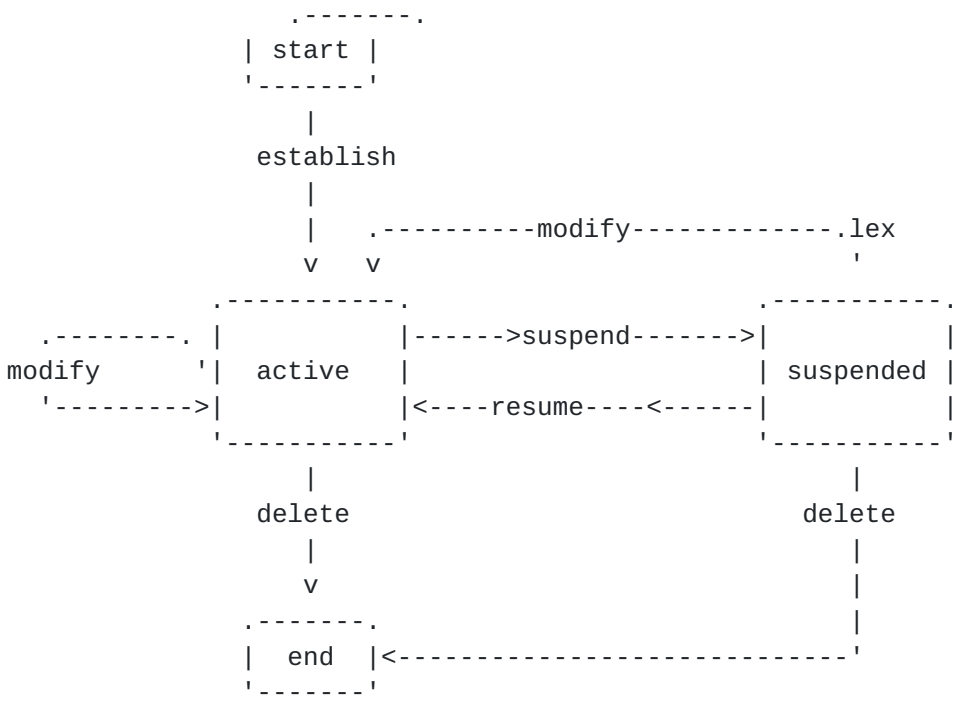


Figure 1: Subscription states at publisher

Of interest in this state machine are the following:

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

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

- A <delete-subscription> request will delete the entire subscription.

2.6. Data Model Trees for Event Notifications

The YANG data models for event notifications are depicted in the following sections.

2.6.1. Data Model Tree for RFC5277 (netconf namespace)

```
module: ietf-5277-netconf
rpcs:
   +---x create-subscription
      +--ro input
         +--ro stream?      string
         +--ro (filter-type)?
         |  +--:(rfc5277)
         |     +--ro filter
         +--ro startTime?   yang:date-and-time
         +--ro stopTime?    yang:date-and-time
```

2.6.2. Data Model Tree for RFC5277 (netmod namespace)

```
module: ietf-5277-netmod
   +--rw netconf
      +--rw streams
         +--rw stream* [name]
            +--rw name                    string
            +--rw description             string
            +--rw replaySupport           boolean
            +--rw replayLogCreationTime   yang:date-and-time
            +--rw replayLogAgedTime       yang:date-and-time
notifications:
   +---n replayComplete
   +---n notificationComplete
```

2.6.3. Data Model for RFC5277-bis Extensions

```
module: ietf-event-notifications
   +--ro streams
   |  +--ro stream*   notif:stream
   +--rw filters
   |  +--rw filter* [filter-id]
   |     +--rw filter-id    filter-id
   |     +--rw (filter-type)?
   |        +--:(rfc5277)
   |           +--rw filter
   +--rw subscription-config {configured-subscriptions}?
   |  +--rw subscription* [subscription-id]
   |     +--rw subscription-id     subscription-id
   |     +--rw stream?             stream
   |     +--rw (filter-type)?
   |     |  +--:(rfc5277)
   |     |  |  +--rw filter
   |     |  +--:(by-reference)
   |     |     +--rw filter-ref?         filter-ref
   |     +--rw startTime?          yang:date-and-time
   |     +--rw stopTime?           yang:date-and-time
   |     +--rw encoding?           encoding
   |     +--rw receivers
   |     |  +--rw receiver* [address]
   |     |     +--rw address     inet:host
   |     |     +--rw port        inet:port-number
   |     |     +--rw protocol?   transport-protocol
   |     +--rw (push-source)?
   |        +--:(interface-originated)
   |        |  +--rw source-interface?   if:interface-ref
   |        +--:(address-originated)
   |           +--rw source-vrf?         uint32
   |           +--rw source-address      inet:ip-address-no-zone
   +--ro subscriptions
      +--ro subscription* [subscription-id]
         +--ro subscription-id            subscription-id
         +--ro configured-subscription?   empty {configured-subscriptions}?
         +--ro subscription-status?       subscription-status
         +--ro stream?                    stream
         +--ro (filter-type)?
         |  +--:(rfc5277)
         |  |  +--ro filter
         |  +--:(by-reference)
         |     +--ro filter-ref?               filter-ref
         +--ro startTime?                 yang:date-and-time
         +--ro stopTime?                  yang:date-and-time
         +--ro encoding?                  encoding
```

```
         +--ro receivers
         |  +--ro receiver* [address]
         |     +--ro address     inet:host
         |     +--ro port        inet:port-number
         |     +--ro protocol?   transport-protocol
         +--ro (push-source)?
            +--:(interface-originated)
            |  +--ro source-interface?          if:interface-ref
            +--:(address-originated)
               +--ro source-vrf?                uint32
               +--ro source-address             inet:ip-address-no-zone
augment /netmod-notif:replayComplete:
   +---- subscription-id?   subscription-id
augment /netmod-notif:notificationComplete:
   +---- subscription-id?   subscription-id
augment /netmod-notif:netconf/netmod-notif:streams:
   +--rw exclude-from-NETCONF-stream?   empty
rpcs:
   +---x establish-subscription
   |  +---w input
   |  |  +---w stream?       stream
   |  |  +---w (filter-type)?
   |  |  |  +--:(rfc5277)
   |  |  |  |  +---w filter
   |  |  |  +--:(by-reference)
   |  |  |     +---w filter-ref?   filter-ref
   |  |  +---w startTime?    yang:date-and-time
   |  |  +---w stopTime?     yang:date-and-time
   |  |  +---w encoding?     encoding
   |  +--ro output
   |     +--ro subscription-result    subscription-result
   |     +--ro (result)?
   |        +--:(success)
   |        |  +--ro subscription-id        subscription-id
   |        +--:(no-success)
   |           +--ro stream?                stream
   |           +--ro (filter-type)?
   |           |  +--:(rfc5277)
   |           |  |  +--ro filter
   |           |  +--:(by-reference)
   |           |     +--ro filter-ref?            filter-ref
   |           +--ro startTime?             yang:date-and-time
   |           +--ro stopTime?              yang:date-and-time
   |           +--ro encoding?              encoding
   +---x modify-subscription
   |  +---w input
   |  |  +---w subscription-id?   subscription-id
   |  |  +---w stream?            stream
```

```
   |  |  +---w (filter-type)?
   |  |  |  +--:(rfc5277)
   |  |  |  |  +---w filter
   |  |  |  +--:(by-reference)
   |  |  |     +---w filter-ref?       filter-ref
   |  |  +---w startTime?         yang:date-and-time
   |  |  +---w stopTime?          yang:date-and-time
   |  |  +---w encoding?          encoding
   |  +--ro output
   |     +--ro subscription-result    subscription-result
   |     +--ro (result)?
   |        +--:(success)
   |        |  +--ro subscription-id        subscription-id
   |        +--:(no-success)
   |           +--ro stream?                stream
   |           +--ro (filter-type)?
   |           |  +--:(rfc5277)
   |           |  |  +--ro filter
   |           |  +--:(by-reference)
   |           |     +--ro filter-ref?            filter-ref
   |           +--ro startTime?             yang:date-and-time
   |           +--ro stopTime?              yang:date-and-time
   |           +--ro encoding?              encoding
   +---x delete-subscription
      +---w input
      |  +---w subscription-id    subscription-id
      +--ro output
         +--ro subscription-result    subscription-result
notifications:
   +---n subscription-started
   |  +--ro subscription-id    subscription-id
   |  +--ro stream?            stream
   |  +--ro (filter-type)?
   |  |  +--:(rfc5277)
   |  |  |  +--ro filter
   |  |  +--:(by-reference)
   |  |     +--ro filter-ref?       filter-ref
   |  +--ro startTime?         yang:date-and-time
   |  +--ro stopTime?          yang:date-and-time
   |  +--ro encoding?          encoding
   +---n subscription-suspended
   |  +--ro subscription-id    subscription-id
   |  +--ro reason?            subscription-susp-reason
   +---n subscription-resumed
   |  +--ro subscription-id    subscription-id
   +---n subscription-modified
   |  +--ro subscription-id    subscription-id
   |  +--ro stream?            stream
```

```
|  +--ro (filter-type)?
|  |  +--:(rfc5277)
|  |  |  +--ro filter
|  |  +--:(by-reference)
|  |     +--ro filter-ref?        filter-ref
|  +--ro startTime?          yang:date-and-time
|  +--ro stopTime?           yang:date-and-time
|  +--ro encoding?           encoding
+---n subscription-terminated
|  +--ro subscription-id     subscription-id
|  +--ro reason?             subscription-term-reason
+---n added-to-subscription
|  +--ro subscription-id     subscription-id
|  +--ro stream?             stream
|  +--ro (filter-type)?
|  |  +--:(rfc5277)
|  |  |  +--ro filter
|  |  +--:(by-reference)
|  |     +--ro filter-ref?        filter-ref
|  +--ro startTime?          yang:date-and-time
|  +--ro stopTime?           yang:date-and-time
|  +--ro encoding?           encoding
+---n removed-from-subscription
   +--ro subscription-id     subscription-id
```

2.7. Creating a Subscription

Editor's note: The following section needs updating. NETCONF mapping will move to a separate document.

This operation is fully defined in [RFC5277]. It allows a subscriber to request the creation of a dynamic subscription. If successful, the subscription remains in effect for the duration of the NETCONF session.

This operation is included in the document for supporting backwards compatibility with [RFC5277] clients. New clients are expected not to use this operation, but establish subscriptions as defined in Section 2.8

2.7.1. Parameters

The input parameters of the operation are:

- stream: An optional parameter that indicates which stream of events is of interest. If not present, events in the default NETCONF stream will be sent.

- filter: An optional parameter that indicates which subset of all possible events is of interest. The format of this parameter is the same as that of the filter parameter in the NETCONF protocol operations. If not present, all events not precluded by other parameters will be sent.

- startTime: An optional parameter used to trigger the replay feature and indicate that the replay should start at the time specified. If startTime is not present, this is not a replay subscription. It is not valid to specify start times that are later than the current time. If the startTime specified is earlier than the log can support, the replay will begin with the earliest available notification. Implementations must support time zones.

- stopTime: An optional parameter used with the optional replay feature to indicate the newest notifications of interest. If stopTime is not present, the notifications will continue until the subscription is terminated. Must be used with and be later than startTime. Implementations must support time zones.

If the server can satisfy the request, it sends a positive acknowledgement.

If the request cannot be completed for any reason, an error is returned along with an error reason. Subscription requests can fail for several reasons, including if a filter with invalid syntax is provided or if the name of a non-existent stream is provided. Other errors include:

- The optional replay feature is requested but the server does not support it

- A stopTime is requested that is earlier than the specified startTime

- A startTime is requested that is later than the current time

2.8. Establishing a Subscription

Editor's note: The following section needs updating. NETCONF mapping will move to a separate document.

This operation is an evolution of the create subscription operation. It allows a subscriber to request the creation of a subscription both via RPC and configuration operations. When invoking the RPC, establish-subscription permits negotiating the subscription terms, changing them dynamically and enabling multiple subscriptions overs a

single NETCONF session (if interleaving [RFC6241] is supported), and canceling subscriptions without terminating the NETCONF session.

The input parameters of the operation are those of create subscription plus:

- filter-ref: filters that have been previously (and separately) configured can be referenced by a subscription. This mechanism enables the reuse of filters.

- encoding: by default, updates are encoded using XML. Other encodings may be supported, such as JSON.

If the NETCONF server cannot satisfy the request, the server sends a negative <subscription-result> element.

If the client has no authorization to establish the subscription, the <subscription-result> indicates an authorization error. If the request is rejected because the server is not able to serve it, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request when it was processed. However, they is no guarantee that subsequent requests with those parameters for this client or others will be accepted. For instance, consider a subscription from [I-D.ietf-netconf-yang-push], which augments the establish-subscription with some additional parameters, including "period".

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

3. Modifying a Subscription

Editor's note: The following section needs updating. NETCONF mapping will move to a separate document.

This operation permits modifying the terms of a subscription previously established. Subscriptions created by configuration cannot be modified. Dynamic subscriptions can be modified one or multiple times. If the server accepts the request, it immediately starts sending events based on the new terms, completely ignoring the previous ones. If the server rejects the request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. The contents of negative responses to modify-subscription requests are the same as in establish subscription requests.

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

Configured subscriptions cannot be modified (or deleted) using RPCs. Instead, configured subscriptions are modified (or deleted) as part of regular configuration operations. Servers MUST reject any attempts to modify (or delete) configured subscriptions via RPC.

The parameters to modify-subscription are those of establish-subscription plus a mandatory subscription-id.

If the NETCONF server can satisfy the request, the server sends a positive subscription-result. This response is like that to an establish-subscription request without the subscription-id, which would be redundant.

If the NETCONF server cannot satisfy the request, the server sends a negative subscription-result. Its contents and semantics are identical to those to an establish-subscription request.

4. Deleting a Subscription

Editor's note: The following section needs updating. NETCONF mapping will move to a separate document.

This operation permits canceling a subscription previously established. Created subscriptions cannot be explicitly deleted. If the server accepts the request, it immediately stops sending events for the subscription. If the server rejects the request, all subscriptions remain as prior to the request. That is, the request has no impact whatsoever. A request may be rejected because the provided subscription identifier is incorrect.

Subscriptions created via RPC can only be deleted via RPC using the same session used for establishment. Configured subscriptions cannot be deleted using RPCs. Instead, configured subscriptions are deleted as part of regular configuration operations. Servers MUST reject any RPC attempt to delete configured subscriptions.

The only parameter to delete-subscription is the identifier of the subscription to delete.

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

If the NETCONF server cannot satisfy the request, the server sends an error-rpc element.

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. This also means configured subscriptions do not support negotiation.

Configured subscriptions can be modified by any configuration client with write rights on the configuration of the subscription. Subscriptions can be modified or terminated 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.

5.1. Creating a Configured Subscription

Configured subscriptions cannot be created via configuration operations. New clients should use the mechanisms described in Section 5.2 for establishing configured subscriptions.

5.2. Establishing a Configured Subscription

Subscriptions can be established using configuration operations against the top-level subtree subscription-config. There are two key differences between RPC and configuration operations for subscription establishment. Firstly, configuration operations do not support negotiation while RPCs do. Secondly, while RPCs mandate that the client establishing the subscription is the only receiver of the notifications, configuration operations permit specifying receivers independent of any tracked subscriber. Immediately after a subscription is successfully established, the server sends to the

receivers a control-plane notification stating the subscription has been established (subscription-started).

Because there is no explicit association with an existing transport session, configured configuration operations require additional parameters to indicate the receivers of the notifications and possibly the source of the notifications (i.e., a specific interface or server address).

For example at subscription establishment, a NETCONF client may send:

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
       xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <subscription-config
            xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
            <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: Establish configured subscription

if the request is accepted, the server would reply:

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

Figure 3: Response to a successful configured subscription establishment

if the request is not accepted because the server cannot serve it, the server may reply:

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

Figure 4: Response to a failed configured subscription establishment

5.3. 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 server 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, added-to-subscription, with the same contents as a modified-subscription) or removed (i.e., removed-from-subscription)

5.4. Deleting a Configured Subscription

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

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <subscription-config
        xmlns:xc="urn:ietf:params:xml:ns:netconf:notification:1.1">
            <subscription xc:operation="delete">
                <subscription-id>
                    1922
                </subscription-id >
           </subscription>
        </subscription-config>
    </edit-config>
</rpc>

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

Figure 5: Deleting a configured subscription

Immediately after a subscription is successfully deleted, the server sends to the receivers 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 NETCONF server sends (asynchronously) the event notifications from the subscribed stream. We refer to these as data plane notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the NETCONF session used to create or establish the subscription. For configured subscriptions, event notifications are sent over the specified connections.

An event notification is sent to the receiver(s) when an event of interest (i.e., meeting the specified filtering criteria) has occurred. An event notification is a complete and well-formed XML document. Note that <notification> is not a Remote Procedure Call (RPC) method but rather the top-level element identifying the one-way message as a notification. Note that event notifications never trigger responses.

The event notification always includes an <eventTime> element. It is the time the event was generated by the event source. This parameter is of type dateTime and compliant to [RFC3339]. Implementations must support time zones.

The event notification also contains notification-specific tagged content, if any. With the exception of <eventTime>, the content of the notification is beyond the scope of this document.

For the encodings other than XML, notifications include an additional XML element so that the notification is a well-formed XML. The element is <notification-contents-{encoding}>, E.g., <notification-contents-json>. That element contains the notification contents in the desired encoding

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

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

Figure 6: Definition of a data plane notification

```
<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 7: Data plane notification

The equivalent using json encoding would be

```
<notification
       xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <notification-contents-json>
       {
          "acme-system:link-failure": {
            "if-name": "so-1/2/3.0",
            "if-admin-status": "up",
            "if-oper-status": "down "
          }
        }
    </notification-contents-json>
 </notification>
```

Figure 8: Data plane notification using JSON encoding

7. Control Plane Notifications

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

Control plane notifications are unlike other notifications in that they are not general-purpose notifications. They cannot be filtered out, and they are delivered only to the receiver of a subscription. They are thus not part of the regular NETCONF event stream. The definition of control plane notifications is distinct from other notifications by making use of a YANG extension tagging them as control plane notification.

Control plane 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. replayComplete

This notification is originally defined in [RFC5277]. It is sent to indicate that all of the replay notifications have been sent and must not be sent for any other reason.

In the case of a subscription without a stop time, after the <replayComplete> notification has been sent, it can be expected that any notifications generated since the start of the subscription creation will be sent, followed by notifications as they arise naturally within the system.

7.2. notificationComplete

This notification is originally defined in [RFC5277]. It is sent to indicate that a subscription, which includes a stop time, has finished passing events.

7.3. 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.

7.4. 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.

7.5. subscription-terminated

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

Clients can terminate via RPC subscriptions established via RPC. In such cases, no subscription-terminated notifications are sent.

7.6. subscription-suspended

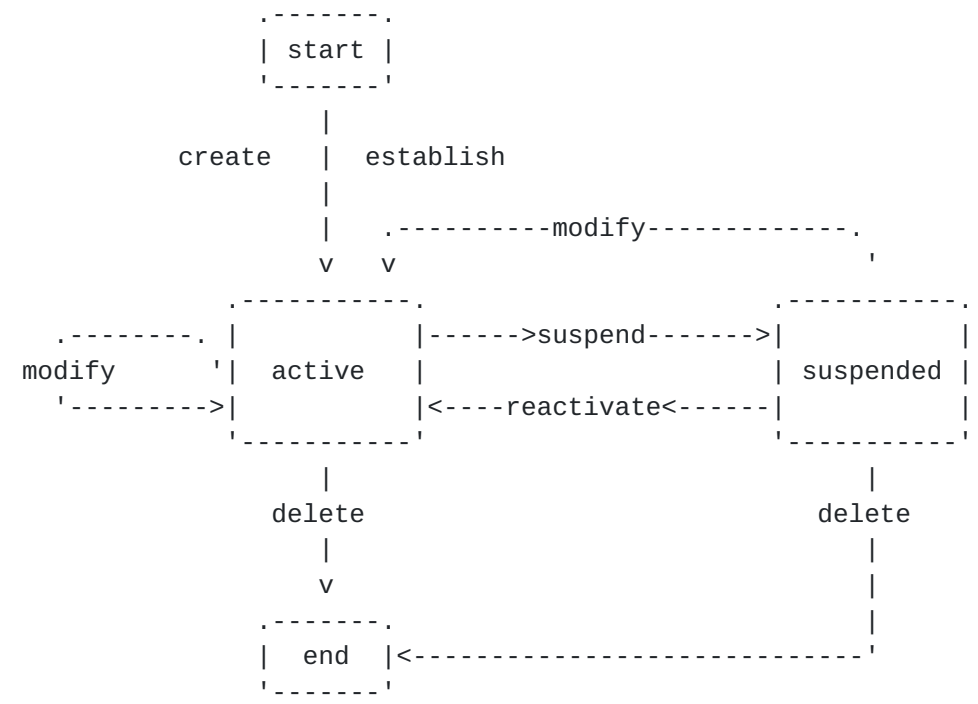
This notification indicates that a server has suspended a subscription. The notification includes the reason for the suspension. A possible reason is the lack of resources to serve it. No further data plane notifications will be sent until the subscription resumes. Suspensions are notified to the subscriber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

7.7. 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).

8. Subscription Management

Below is the state machine for the server. It is important to note that a subscription does not exist at the agent until it is accepted and made active.



Of interest in this state machine are the following:

- Successful <create-subscription>, <establish-subscription> or <modify-subscription> actions must put the subscription into an active state.

- Failed <modify-subscription> actions will leave the subscription in its previous state, with no visible change to any notifications.

- A <delete-subscription> action will delete the entire subscription.

9. Data Models for Event Notifications

9.1. Data Model for RFC5277 (netmod namespace)

```
<CODE BEGINS>
file "ietf-5277-netmod@2016-06-15.yang"
module ietf-5277-netmod {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-5277-netmod";
    // TODO: examples in the draft consider the namespace below
    //       which follows the namespace format in 5277
    // "urn:ietf:params:xml:ns:netmod:notification";
  prefix netmod-notif;

  import ietf-yang-types {
    prefix yang;
  }

  organization "IETF";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>


     WG Chair: Juergen Schoenwaelder
               <mailto:j.schoenwaelder@jacobs-university.de>
     WG Chair: Tom Nadeau
                <mailto:tnadeau@lucidvision.com>

     Editor:   Alberto Gonzalez Prieto
               <mailto:albertgo@cisco.com>

     Editor:   Alexander Clemm
               <mailto:alex@cisco.com>

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

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

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

  description
    "Model for RPC in RFC 5277: NETCONF Event Notifications";

  revision 2016-06-15 {
```

```
      description
        "Model for data nodes and notifications in RFC 5277:
         NETCONF Event Notifications";
      reference
        "RFC 5277: NETCONF Event Notifications";
    }


    /*
     * EXTENSIONS
     */

    extension control-plane-notif {
      description
        "This statement applies only to notifications.
         It indicates that the notification is a control-plane
         notification and therefore it does not generate an
         event stream.";
    }


    /*
     * DATA NODES
     */
    container netconf {
      description
        "Netconf container as defined in RFC 5277.";
      reference
        "RFC 5277: NETCONF Event Notifications";

      container streams {
        // TODO: should be config false?. That breaks the leafref
        // from the config subscriptions config false;
        description
          "The container with the set of available event streams.";
        reference
          "RFC 5277: NETCONF Event Notifications";

        list stream {
          must "1 = count(./name == 'NETCONF')" {
            description
              "The list must contain a NETCONF stream.
               A NETCONF server implementation supporting the
               notification capability MUST support the
               'NETCONF' notification event stream. This stream
               contains all NETCONF XML event notifications supported
               by the NETCONF server.
               The exact string 'NETCONF' is used during the
```

```
          advertisement of stream support during the <get>
          operation on <streams> and during the
          <create-subscription> operation.";
      }
      key "name";
      description
        "The list available event streams.";
      leaf name {
        type string;
        description
          "The name of the event stream.
           If this is the default NETCONF stream, this must have
           the value 'NETCONF'.";
      }
      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 replaySupport {
        type boolean;
        mandatory true;
        description
          "An indication of whether or not event replay is
           available on this stream.";
      }
      leaf replayLogCreationTime {
        when "../replaySupport" {
          description
            "This object MUST be present if replay is supported.";
        }
        type yang:date-and-time;
        mandatory true;
        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 replayLogAgedTime {
        when "current()/../replaySupport" {
          description
            "This object MUST be present if replay is supported
```

```
                and any notifications have been aged out of the log.";
            }
            type yang:date-and-time;
            mandatory true;
            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.";
          }
        } // list stream
      } // container streams
    } // container netconf


    /*
     * NOTIFICATIONS
     */

    notification replayComplete {
      netmod-notif:control-plane-notif;
      description
        "This notification is sent to signal the end of a replay
         portion of a subscription.";
    }

    notification notificationComplete {
      netmod-notif:control-plane-notif;
      description
        "This notification is sent to signal the end of a notification
         subscription.  It is sent in the case that stopTime was
         specified during the creation of the subscription.";
    }
  }

  <CODE ENDS>
```

9.2. Data Model for RFC5277 (netconf namespace)

```
  <CODE BEGINS>
  file "ietf-5277-netconf@2016-06-15.yang"
  module ietf-5277-netconf {
    namespace
      "urn:ietf:params:xml:ns:yang:ietf-5277-netconf";
      // TODO: examples in the draft consider the namespace below
      //       which follows the namespace format in 5277
      // "urn:ietf:params:xml:ns:netconf:notification:1.0";
    prefix notif;
```

```
     import ietf-yang-types {
       prefix yang;
     }

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

        WG Chair: Mahesh Jethanandani
                  <mailto:mjethanandani@gmail.com>

        WG Chair: Mehmet Ersue
                  <mailto:mehmet.ersue@nokia.com>

        Editor:   Alberto Gonzalez Prieto
                  <mailto:albertgo@cisco.com>

        Editor:   Alexander Clemm
                  <mailto:alex@cisco.com>

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

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

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

     description
       "Model for RPCs and Notifications in RFC 5277: NETCONF Event
        Notifications";

     revision 2016-06-15 {
       description
         "Model for RPC in RFC 5277: NETCONF Event Notifications";
       reference
         "RFC 5277: NETCONF Event Notifications";
     }

     /*
      * IDENTITIES
      */

     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.";
      }

      /*
       * TYPEDEFS
       */

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

      /*
       * GROUPINGS
       */

      grouping base-filter {
        description
          "This grouping defines the base for filters for
           notification events.
           It includes the filter defined in 5277 and
           it enables extending filtering to other
           types of 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 rfc5277 {
            anyxml filter {
              description
                "Filter per RFC 5277. Notification filter.
                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-info-5277 {
      description
        "This grouping describes the information in a 5277
         subscription.";
      leaf stream {
        type stream;
        default "NETCONF";
        description
          "Indicates which stream of events is of interest.
           If not present, events in the default NETCONF stream
           will be sent.";
      }
      uses base-filter;
      leaf startTime {
        type yang:date-and-time;
        description
          "Used to trigger the replay feature
           and indicate that the replay should start at the time
           specified.  If <startTime> is not present, this is not a
           replay subscription.  It is not valid to specify start
           times that are later than the current time.  If the
           <startTime> specified is earlier than the log can support,
           the replay will begin with the earliest available
           notification.  This parameter is of type dateTime and
           compliant to [RFC3339].  Implementations must
           support time zones.";
      }
      leaf stopTime {
        type yang:date-and-time;
        must "current()  > ../startTime" {
          description
            "stopTime must be used with and be later than <startTime>";
        }
        description
          "Used with the optional replay feature to indicate the
           newest notifications of interest.  If <stopTime> is
```

```
            not present, the notifications will continue until the
            subscription is terminated.  Must be used with and be
            later than <startTime>.  Values of <stopTime> in the
            future are valid.  This parameter is of type dateTime and
            compliant to [RFC3339].  Implementations must support time
            zones.";
        }
      }


      /*
       * RPCs
       */

      rpc create-subscription {
        description
          "This operation initiates an event notification subscription
           that will send asynchronous event notifications to the
           initiator of the command until the subscription terminates.";
        input {
          uses subscription-info-5277;
        }
      }
    }

    <CODE ENDS>
```

9.3. Data Model for RFC5277-bis Extensions

```
    <CODE BEGINS>
    file "ietf-event-notifications@2016-06-15.yang"
    module ietf-event-notifications {
      namespace
        "urn:ietf:params:xml:ns:yang:ietf-event-notifications";
        // TODO: examples in the draft consider the namespace below
        //       which follows the namespace format in 5277
        // "urn:ietf:params:xml:ns:netconf:notification:1.1";
      prefix notif-bis;


      import ietf-inet-types {
        prefix inet;
      }
      import ietf-5277-netmod {
        prefix netmod-notif;
      }
      import ietf-5277-netconf {
        prefix notif;
```

```
    }
    import ietf-interfaces {
      prefix if;
    }

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

       WG Chair: Mahesh Jethanandani
                 <mailto:mjethanandani@gmail.com>

       WG Chair: Mehmet Ersue
                 <mailto:mehmet.ersue@nokia.com>

       Editor:   Alberto Gonzalez Prieto
                 <mailto:albertgo@cisco.com>

       Editor:   Alexander Clemm
                 <mailto:alex@cisco.com>

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

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

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

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

    revision 2016-06-15 {
      description
        "Initial version. Model for NETCONF Notifications (bis)";
      reference
        "RFC XXXX: NETCONF Event Notifications";
    }


    /*
     * 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.";
    }

    /*
     * IDENTITIES
     */

    /* 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.";
    }

    identity error-no-such-subscription {
      base error;
      description
        "A subscription with the requested subscription ID
         does not exist.";
    }

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

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

    identity error-configured-subscription {
      base error;
      description
        "Cannot apply RPC to a configured subscription, i.e.
         to a subscription that was not established via RPC.";
    }

    identity error-other {
      base error;
      description
        "An unspecified error has occurred (catch all).";
    }

    /* 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 push server
         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 subscription-errors {
      description
        "Base identity for subscription error status.
         This identity is not to be confused with error return
         codes for RPCs";
    }

    identity internal-error {
      base subscription-errors;
      description
        "Subscription failures caused by server internal error.";
    }

    identity no-resources {
      base subscription-errors;
      description
        "Lack of resources, e.g. CPU, memory, bandwidth";
    }

    identity subscription-deleted {
      base subscription-errors;
      description
        "The subscription was terminated because the subscription
         was deleted.";
    }

    identity other {
      base subscription-errors;
      description
        "Fallback reason - any other reason";
    }

    /* 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-term-reason {
      type identityref {
        base subscription-errors;
      }
      description
        "Reason for a server to terminate a subscription.";
    }

    typedef subscription-susp-reason {
      type identityref {
        base subscription-errors;
      }
      description
        "Reason for a server to suspend a subscription.";
    }

    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 push-source {
      type enumeration {
        enum "interface-originated" {
          description
            "Notifications will be sent from a specific interface on a
             NETCONF server";
        }
```

```
        enum "address-originated" {
          description
            "Notifications will be sent from a specific address on a
             NETCONF server";
        }
      }
      description
        "Specifies from where notifications will be sourced when
         being sent by the NETCONF server.";
    }

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

    /*
     * GROUPINGS
     */

    grouping subscription-info {
      description
        "This grouping describes basic information concerning a
         subscription.";
      uses notif:subscription-info-5277 {
        augment "filter-type" {
          description
            "Post-5277 subscriptions allow references to existing
             filters";
          case by-reference {
            description
              "Incorporate a filter that has been configured
               separately.";
            leaf filter-ref {
              type filter-ref;
              description
                "References filter which is associated with the
                 subscription.";
            }
          }
        }
      }
      leaf encoding {
        type encoding;
        default "encode-xml";
```

```
        description
          "The type of encoding for the subscribed data.
           Default is XML";
      }
    }

    grouping push-source-info {
      description
        "Defines the sender source from which notifications
         for a configured subscription are sent.";
      choice push-source {
        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 uint32 {
              range "16..1048574";
            }
            description
              "Label of the vrf.";
          }
          leaf source-address {
            type inet:ip-address-no-zone;
            mandatory true;
            description
              "The source address for the notifications.";
          }
        }
      }
    }

    grouping receiver-info {
      description
        "Defines where and how to deliver notifications for a
```

```
         configured subscription.  This includes
         specifying the receiver, as well as defining
         any network and transport aspects when sending of
         notifications occurs outside of Netconf.";
      container receivers {
        description
          "Set of receivers in a subscription.";
        list receiver {
          key "address";
          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.";
          }
        }
      }
    }

    grouping subscription-response {
      description
        "Defines the output to the rpc's establish-subscription
         and modify-subscription.";
      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 success {
          description
            "This case is used when the subscription request
             was successful and a subscription was created/modified
             as a result";
          leaf subscription-id {
            type subscription-id;
            mandatory true;
            description
              "Identifier used for this subscription.";
          }
        }
        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 subscription-info;
        }
      }
    }

    /*
     * 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 by virtue of a delete-subscription request.
         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 create-subscription request.";
      input {
        uses subscription-info;
      }
      output {
        uses subscription-response;
      }
    }

    rpc modify-subscription {
      description
        "This RPC allows a subscriber to modify a subscription
         that was previously created using create-subscription.
         If successful, the subscription
         remains in effect for the duration of the subscriber's
         association with the publisher, or until the subscription
         is terminated by virtue of a delete-subscription request.
         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 settings
         that would have a high likelihood of succeeding in a
         subsequent modify-subscription request.";
      input {
        leaf subscription-id {
          type subscription-id;
          description
            "Identifier to use for this subscription.";
        }
        uses subscription-info;
      }
      output {
        uses subscription-response;
      }
    }

    rpc delete-subscription {
      description
        "This RPC allows a subscriber to delete a subscription that
         was previously created using create-subscription.";
      input {
        leaf subscription-id {
          type subscription-id;
          mandatory true;
```

```
          description
            "Identifier of the subscription that is to be deleted.
             Only subscriptions that were created using
             create-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 subscription-started {
      netmod-notif:control-plane-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 subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      uses subscription-info;
    }

    notification subscription-suspended {
      netmod-notif:control-plane-notif;
      description
        "This notification indicates that a suspension of the
         subscription by the server has occurred.  No further
         notifications will be sent until subscription
         resumes.
         This notification shall only be sent to receivers
         of a subscription; it does not constitute a general-purpose
```

```
         notification.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      leaf reason {
        type subscription-susp-reason;
        description
          "Provides a reason for why the subscription was
           suspended.";
      }
    }

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

    notification subscription-modified {
      netmod-notif:control-plane-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 subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      uses subscription-info;
    }

    notification subscription-terminated {
      netmod-notif:control-plane-notif;
      description
```

```
        "This notification indicates that a subscription has been
         terminated.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      leaf reason {
        type subscription-term-reason;
        description
          "Provides a reason for why the subscription was
           terminated.";
      }
    }

    notification added-to-subscription {
      netmod-notif:control-plane-notif;
      description
        "This notification is sent to a receiver when it has been
         added to an existing subscription.
         Note that if the receiver is added when the subscription
         is created, it will receive a subscription-started
         notification and no added-to-subscription.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      uses subscription-info;
    }

    notification removed-from-subscription {
      netmod-notif:control-plane-notif;
      description
        "This notification is sent to a receiver when it has been
         removed from an existing subscription.
         Note that if the subscription is terminated, the receiver
         will receive a subscription-terminated notification
         and no removed-from-subscription.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
    }
```

```
    augment /netmod-notif:replayComplete {
      description
        "Augmenting the definition in RFC-5277 to include the
         subscription identifier defined in 5277-bis";
      leaf subscription-id {
        type subscription-id;
        description
          "This references the affected subscription.
           Not present for created subscriptions for backwards
           compatibility.";
      }
    }

    augment /netmod-notif:notificationComplete {
      description
        "Augmenting the definition in RFC-5277 to include the
         subscription identifier defined in 5277-bis";
      leaf subscription-id {
        type subscription-id;
        description
          "This references the affected subscription.
           Not present for created subscriptions for backwards
           compatibility.";
      }
    }


    /*
     * 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 notif: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 "filter-id";
        description
          "A list of configurable filters that can be applied to
           subscriptions.";
        leaf filter-id {
          type filter-id;
          description
            "An identifier to differentiate between filters.";
        }
        uses notif: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 "subscription-id";
        description
          "Content of a subscription.";
        leaf subscription-id {
          type subscription-id;
          description
            "Identifier to use for this subscription.";
        }
        uses subscription-info;
        uses receiver-info {
          if-feature "configured-subscriptions";
        }
        uses push-source-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, e.g. create-subscription, delete-subscription,
```

```
         and modify-subscription, as well as subscriptions that
         have been established via configuration.";
      list subscription {
        key "subscription-id";
        config false;
        description
          "Content of a subscription.
           Subscriptions can be created using a control channel
           or RPC, or be established through configuration.";
        leaf subscription-id {
          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.";
        }

        leaf subscription-status {
          type subscription-status;
          description
            "The status of the subscription.";
        }
        uses subscription-info;
        uses receiver-info {
          if-feature "configured-subscriptions";
        }
        uses push-source-info {
          if-feature "configured-subscriptions";
        }
      }
    }

    augment /netmod-notif:netconf/netmod-notif:streams {
      description
        "Augmenting the definition in RFC-5277 to so that streams
         can opt out the default stream.";
      leaf exclude-from-NETCONF-stream {
          type empty;
          description
            "Indicates that the stream should not be part of the
             default stream.";
```

```
      }
    }

  }

  <CODE ENDS>
```

10. Backwards Compatibility

Capabilities are advertised in messages sent by each peer during session establishment [RFC6241]. Servers supporting the features in this document must advertise both capabilities "urn:ietf:params:netconf:capability:notification:1.0" and "urn:ietf:params:netconf:capability:notification:1.1".

An example of a hello message by a server during session establishment would be:

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>
            urn:ietf:params:xml:ns:netconf:base:1.0
        </capability>
        <capability>
            urn:ietf:params:netconf:capability:startup:1.0
        </capability>
        <capability>
            urn:ietf:params:netconf:capability:notification:1.0
        </capability>
        <capability>
            urn:ietf:params:netconf:capability:notification:1.1
        </capability>
    </capabilities>
    <session-id>4</session-id>
</hello>
```

Figure 9: Hello message

Clients that only support [RFC5277] recognize capability "urn:ietf:params:netconf:capability:notification:1.0" and ignore capability "urn:ietf:params:netconf:capability:notification:1.1". This allows them interacting with the server as per [RFC5277]. Clients that support the features in this document recognize both capabilities. This allows them interacting with the server as per this document.

Note that to support backwards compatibility, the yang models in this document include two types of naming conventions. That used in

[RFC5277], e.g., replayComplete; and that commonly used in yang models, e.g., subscription-started.

11. Security Considerations

The security considerations from the base NETCONF document [RFC6241] also apply to the notification capability.

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

A secure transport must be used and the server must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of NETCONF content involved. When a <get> is received that refers to the content defined in this memo, clients should only be able to view the content for which they have sufficient privileges. <create-subscriptiont> and <establish-subscriptiont> 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> that different users are able to subscribe to.

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

If a malicious or buggy NETCONF client sends a number of <create-subscription> requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions can be terminated by terminating the suspect underlying NETCONF sessions using the <kill-session> operation. If the client uses <establish-subscription>, the server 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. Clients 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.

12. Issues that are currently being worked and resolved

12.1. Unresolved and yet-to-be addressed issues

EN1 - Definition of basic set of Stream types. What streams are provided and what do they contain (includes default 5277 stream).

EN2 - Clarify interplay between filter definitions and different streams. Includes information in subtrees of event payloads.

EN3 - Mechanisms for diagnostics, e.g. deal with dropped updates, monitoring when they occur, etc

EN4 - How to allow for seamless integration with non-standard encodings and transports (like GPB/GRPC). Specify requirements encoding and transport must meet, provide examples.

EN5 - Along with Netconf-notif, should this draft obsolete 5277 or be in parallel with it?

EN6 - Stream discovery. Are adjustments needed for maximal transport independence?

EN7 - Detecting loss of a sequential update notification, and mechanisms to resend. Implications to transports must be thought through.

EN8 - Should we have a mandatory transport?

12.2. Agreement in principal

EN9 - Multiple receivers per Configured Subscription is ok.

EN10 - Replay support will be provided for selected stream types (modify vs. delete)

EN11 - Required layering security requirements/considerations will be added into the YANG model for Configured Subscriptions. It will be up to the transport to meet these requirements.

EN12 - Test-only option for a subscription is desired. But it still needs to be defined.

EN13 - RFC6241 Subtree-filter definition in 5277bis cannot apply to elements of an event. Must explicitly define how 6241 doesn't apply filtering within a 5277bis event.

EN14 - Ensure that Configured Subscriptions are fully defined in YANG model.

12.3. Resolved Issues

EN15 - Term for Dynamic and Static Subscriptions (move to "Configured")

12.4. Editorial To-Dos

Update examples. Examples are not in synch with the current model.

Remove overlap with transport specifics, specifically NETCONF. The current draft revision is "self-contained" and assumes NETCONF as a transport. It does not yet reflect the breakout of transport mappings into a separate draft.

13. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Yang Geng, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, and Guangying Zheng.

14. References

14.1. 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>.

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

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

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

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

14.2. Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-13, April 2016.

[I-D.ietf-netconf-yang-push]
Clemm, Alexander., Gonzalez Prieto, Alberto., Voit, Eric., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", June 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Authors' Addresses

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Alexander Clemm
Cisco Systems

Email: alex@cisco.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Sharon Chisholm
Ciena

Email: schishol@ciena.com

Hector Trevino
Cisco Systems

Email: htrevino@cisco.com