Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 1, 2017

Pre-release version

A. Clemm
Sympotech
E. Voit
A. Gonzalez Prieto
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Bierman
YumaWorks
B. Lengyel
Ericsson

October 28, 2016

Subscribing to YANG datastore push updates
draft-ietf-netconf-yang-push-04

Abstract

This document defines a subscription and push mechanism for YANG datastores. This mechanism allows subscriber applications to request updates from a YANG datastore, which are then pushed by the publisher to a receiver per a subscription policy, without requiring additional subscriber requests.

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 May 1, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

YANG [RFC7950] was originally designed for the Netconf protocol [RFC6241] which focused on configuration data. However, YANG can be used to model both configuration and operational data. It is therefore reasonable to expect YANG datastores will increasingly be used to support applications that care about about both.

For example, service assurance applications will need to be aware of any remote updates to configuration and operational objects. Rapid awareness of object changes will enable such things as validating and maintaining cross-network integrity and consistency, or monitoring state and key performance indicators of remote devices.

Traditional approaches to remote visibility have been built on polling. With polling, data is periodically explicitly retrieved by a client from a server to stay up-to-date. However, there are issues associated with polling-based management:

- It introduces additional load on network, devices, and applications. Each polling cycle requires a separate yet arguably redundant request that results in an interrupt, requires parsing, consumes bandwidth.

- It lacks robustness. Polling cycles may be missed, requests may be delayed or get lost, often particularly in cases when the network is under stress and hence exactly when the need for the data is the greatest.

- Data may be difficult to calibrate and compare. Polling requests may undergo slight fluctuations, resulting in intervals of different lengths which makes data hard to compare. Likewise, pollers may have difficulty issuing requests that reach all devices at the same time, resulting in offset polling intervals which again make data hard to compare.

A more effective alternative to polling is when an application can request to be automatically updated on current relevant content of a datastore. If such a request is accepted, interesting updates will subsequently be pushed from that datastore.

Dependence on polling-based management is typically considered an important shortcoming of applications that rely on MIBs polled using SNMP [RFC1157]. However, without a provision to support a push-based alternative, there is no reason to believe that management applications that operate on YANG datastores will be any more effective, as they would follow the same request/response pattern.

While YANG allows the definition of push notifications, such notifications generally indicate the occurrence of certain well-specified event conditions, such as the onset of an alarm condition or the occurrence of an error. A capability to subscribe to and deliver such pre-defined event notifications has been defined in [RFC5277]. In addition, configuration change notifications have been defined in [RFC6470]. These change notifications pertain only to configuration information, not to operational state, and convey the root of the subtree to which changes were applied along with the edits, but not the modified data nodes and their values. Furthermore, while delivery of updates using notifications is a viable option, some applications desire the ability to stream updates using other transports.

Accordingly, there is a need for a service that allows applications to dynamically subscribe to updates of a YANG datastore and that allows the publisher to push those updates, possibly using one of several delivery mechanisms. Additionally, support for subscriptions configured directly on the publisher are also useful when dynamic signaling is undesirable or unsupported. The requirements for such a service are documented in [RFC7923].

This document proposes a solution. The solution builds on top of the Netconf Event Model [I-D.ietf-netconf-5277bis] which defines a mechanism for the management of event subscriptions. At its core, the solution defined here introduces a new set of event streams which maybe subscribed, introduces datastore push update mechanisms, and provides extensions to the event subscription model. The document also includes YANG data model augmentations which extend the model and RPCs defined within [I-D.ietf-netconf-5277bis].

Key capabilities worth highlighting include:

- An extension to event subscription mechanisms allowing clients to subscribe to event streams containing automatic datastore updates. The subscription allows clients to specify which data they are interested in, what types of updates (e.g. create, delete, modify), and to provide optional filters with criteria that data must meet for updates to be sent. Furthermore, subscriptions can specify a policy that directs when updates are provided. For

example, a client may request to be updated periodically in certain intervals, or whenever data changes occur.

- Format and contents of the YANG push updates themselves.

- The ability for a publisher to push back on requested subscription parameters. Because not every publisher may support every requested update policy for every piece of data, it is necessary for a publisher to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to negotiate push update subscription parameters. For example, some publishers may have a lower limit to the period with which they can send updates, or they may not support on-change updates for every piece of data.

- Subscription parameters which allow the specification of QoS extensions to address prioritization between independent streams of updates.

2. Definitions and Acronyms

Many of the terms in this document are defined in [I-D.ietf-netconf-5277bis]. Please see that document for these definitions.

Data node: An instance of management information in a YANG datastore.

Data node update: A data item containing the current value/property of a Data node at the time the data node update was created.

Data record: A record containing a set of one or more data node instances and their associated values.

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

Datastream: A continuous stream of data records, each including a set of updates, i.e. data node instances and their associated values.

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

Push-update stream: A conceptual data stream of a datastore that streams the entire datastore contents continuously and perpetually.

Update: A data item containing the current value of a data node.

Update notification: An Event Notification including those data node update(s) to be pushed in order to meet the obligations of a single Subscription. All included data node updates must reflect the state of a Datastore at a snapshot in time.

Update record: A representation of a data node update as a data record. An update record can be included as part of an update stream. It can also be logged for retrieval. In general, an update record will include the value/property of a data node. It may also include information about the type of data node update, i.e. whether the data node was modified/updated, or newly created, or deleted.

Update trigger: A mechanism, as specified by a Subscription Policy, that determines when a data node update is to be communicated. (e.g., a change trigger, invoked when the value of a data node changes or a data node is created or deleted, or a time trigger, invoked after the laps of a periodic time interval.)

YANG object filter: A filter that contains evaluation criteria which are evaluated against YANG objects of a subscription. An update is only published if the object meets the specified filter criteria.

YANG-Push: The subscription and push mechanism for YANG datastores that is specified in this document.

3. Solution Overview

This document specifies a solution for a push update subscription service. This solution supports the dynamic as well as configured subscriptions to information updates from YANG datastores. A subscription might target exposed operational and/or configuration YANG objects on a device. YANG objects are subsequently pushed from the publisher to the receiver per the terms of the subscription.

3.1. Subscription Model

YANG-push subscriptions are defined using a data model that is itself defined in YANG. This model augments the event subscription model defined in [I-D.ietf-netconf-5277bis] and introduces several new parameters that allow subscribers to specify what to include in an update notification and what triggers such an update notification.

The subscription model assumes the presence of one or more conceptual perpetual datastreams of continuous subscribable YANG updates. There are several datastreams with predefined semantics, such as the stream of updates of all operational data or the stream of updates of all config data. In addition, it is possible to define custom streams with customizable semantics. The model includes the list of update

datastreams that are supported by a system and available for subscription.

The subscription model augments the [I-D.ietf-netconf-5277bis] subscription model with a set of parameters:

- Anydata encoding for periodic and on-change push updates.

- A subscription policy definition regarding the update trigger when to send new update notifications.

  * For periodic subscriptions, the trigger is defined by two parameters that defines the interval with which updates are to be pushed. These parameters are the period/interval of reporting duration, and an anchor time which can be used to calculate at which times updates needs to be assembled and sent.

  * EDITOR'S NOTE: A possible option to discuss concerns the introduction of an additional parameter "changes-only" for periodic subscription. Including this flag would results in sending at the end of each period an update containing only changes since the last update (i.e. a change-update as in the case of an on-change subscription), not a full snapshot of the subscribed information. Such an option might be interesting in case of data that is largely static and bandwidth-constrained environments.

  * For on-change subscriptions, the trigger occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that can be guided by additional parameters. Please refer also to Section 3.3.

    + One parameter specifies the dampening period, i.e. the interval that must pass before a successive update notification for the same Subscription is sent. Note that the dampening period applies to the set of all data nodes within a single subscription. This means that on the first change of an object, an update notification containing that object is sent either immediately or at the end of a dampening period already in effect.

    + Another parameter allows to restrict the types of changes for which updates are sent (changes to object values, object creation or deletion events). It is conceivable to augment the data model with additional parameters in the future to specify even more refined policies, such as parameters that

specify the magnitude of a change that must occur before an update is triggered.

+ A third parameter specifies whether or not a complete update with all the subscribed data should be sent at the beginning of a subscription to facilitate synchronization and establish the frame of reference for subsequent updates.

o Optionally, a filter, or set of filters, describing the subset of data node updates that are of interest to the subscriber. The publisher must only send to the subscriber those data node updates that can traverse applied filter(s). The absence of a filter indicates that all data items from the stream are of interest to the subscriber and all data records must be sent in their entirety to the subscriber. The following types of filters are supported: subtree filters, with the same semantics as defined in [RFC6241][RFC6241], and XPath filters. Additional filter types can be added through augmentations. Filters can be specified "inline" as part of the subscription, or can be configured separately and referenced by a subscription, in order to facilitate reuse of complex filters.

The subscription data model is specified as part of the YANG data model described later in this specification. It is conceivable that additional subscription parameters might be added in the future. This can be accomplished through augmentation of the subscription data model.

3.2. Negotiation of Subscription Policies

Dynamic subscriptions must support a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is limited to a single pair of subscription request and response messages. For negative response messages, the publisher SHOULD include in the returned error what subscription parameters would have been accepted for the request. The returned acceptable parameters constitute suggestions that, when followed, increase the likelihood of success for subsequent requests. However, there are no guarantee that subsequent requests for this subscriber will in fact be accepted.

A subscription request might be declined based on publisher's assessment that it may be unable to provide a filtered update notification stream that would meet the terms of the establish-subscription request.

In case a subscriber requests an encoding other than XML, and this encoding is not supported by the publisher, the publisher simply indicates in the response that the encoding is not supported.

A subscription negotiation capability has been introduced as part of the NETCONF Event Notifications model. However, the ability to negotiate subscriptions is of particular importance in conjunction with push updates, as publisher implementations may have limitations with regards to what updates can be generated and at what velocity.

3.3. On-Change Considerations

On-change subscriptions allow subscribers to subscribe to updates whenever changes to objects occur. As such, on-change subscriptions are of particular interest for data that changes relatively infrequently, yet that require applications to be notified with minimal delay when changes do occur.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Specifically, on-change subscriptions may involve a notion of state to see if a change occurred between past and current state, or the ability to tap into changes as they occur in the underlying system. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

When an on-change subscription is requested for a datastream with a given subtree filter, where not all objects support on-change update triggers, the subscription request MUST be rejected. As a result, on-change subscription requests will tend to be directed at very specific, targeted subtrees with only few objects.

Any updates for an on-change subscription will include only objects for which a change was detected. To avoid flooding receivers with repeated updates for fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update for a given object is sent, no other updates for this particular object are sent until the end of the dampening period. Values sent at the end of the dampening period are the values current when that dampening period expires. In addition, updates include information about objects that were deleted and ones that were newly created.

On-change subscriptions can be refined to let users subscribe only to certain types of changes, for example, only to object creations and deletions, but not to modifications of object values.

Additional refinements are conceivable. For example, in order to avoid sending updates on objects whose values undergo only a

negligible change, additional parameters might be added to an on-change subscription specifying a YANG object filter that states how large or "significant" a change has to be before an update is sent. A simple policy is a "delta-policy" that states, for integer-valued data nodes, the minimum difference between the current value and the value that was last reported that triggers an update. Also more sophisticated policies are conceivable, such as policies specified in percentage terms or policies that take into account the rate of change. While not specified as part of this draft, such policies can be accommodated by augmenting the subscription data model accordingly.

3.4. Data Encodings

Subscribed data is encoded in either XML or JSON format. A publisher MUST support XML encoding and MAY support JSON encoding.

It is conceivable that additional encodings may be supported as options in the future. This can be accomplished by augmenting the subscription data model with additional identity statements used to refer to requested encodings.

3.4.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been simply retrieved using a get operation and is encoded in the same way. XML encoding rules for data nodes are defined in [RFC7950]. JSON encoding rules are defined in [RFC7951]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

3.4.2. On-Change Subscriptions

In an on-change subscription, updates need to indicate not only values of changed data nodes but also the types of changes that occurred since the last update, such as whether data nodes were newly created since the last update or whether they were merely modified, as well as which data nodes were deleted.

Encoding rules for data in on-change updates correspond to how data would be encoded in a YANG-patch operation as specified in [I-D.ietf-netconf-yang-patch]. The "YANG-patch" would in this case be applied to the earlier state reported by the preceding update, to result in the now-current state of YANG data. Of course, contrary to a YANG-patch operation, the data is sent from the publisher to the receiver and is not restricted to configuration data.

3.5. YANG object filters

Subscriptions can specify filters for subscribed data. The following filters are supported:

- subtree-filter: A subtree filter specifies a subtree that the subscription refers to. When specified, updates will only concern data nodes from this subtree. Syntax and semantics correspond to that specified for [RFC6241] section 6.

- xpath-filter: An XPath filter specifies an XPath expression applied to the data in an update, assuming XML-encoded data.

Only a single filter can be applied to a subscription at a time.

It is conceivable for implementations to support other filters. For example, an on-change filter might specify that changes in values should be sent only when the magnitude of the change since previous updates exceeds a certain threshold. It is possible to augment the subscription data model with additional filter types.

3.6. Push Data Stream and Transport Mapping

Pushing data based on a subscription could be considered analogous to a response to a data retrieval request, e.g. a "get" request. However, contrary to such a request, multiple responses to the same request may get sent over a longer period of time.

An applicable mechanism is that of a notification. There are however some specifics that need to be considered. Contrary to other notifications that are associated with alarms and unexpected event occurrences, update notifications are solicited, i.e. tied to a particular subscription which triggered the notification.

A push update notification contains several parameters:

- A subscription correlator, referencing the name of the subscription on whose behalf the notification is sent.

- Data nodes containing a representation of the datastore subtree(s) containing the updates. In all cases, the subtree(s) are filtered per access control rules to contain only data that the subscriber is authorized to see. For on-change subscriptions, the subtree may only contain the data nodes which have changed since the start of the previous dampening interval.

This document introduces two generic notifications: "push-update" and "push-change-update". Those notifications may be encapsulated on a

transport (e.g. Netconf notifications and HTTP) to carry data records with updates of datastore contents as specified by a subscription. It is possible also map notifications to other transports and encodings and use the same subscription model; however, the definition of such mappings is outside the scope of this document.

A push-update notification defines a complete update of the datastore per the terms of a subscription. This type of notification is used for continuous updates of periodic subscriptions. A push-update notification can also used be for the on-change subscriptions in two cases. First it will be used as the initial push-update if there is a need to synchronize the receiver at the start of a new subscription. It also may be sent if the publisher later chooses to resynch a previously synched on-change subscription. The push-update record contains a data snippet that contains an instantiated subtree with the subscribed contents. The content of the update notification is equivalent to the contents that would be obtained had the same data been explicitly retrieved using e.g. a Netconf "get"-operation, with the same filters applied.

The contents of the notification conceptually represents the union of all data nodes in the yang modules supported by the publisher. However, in a YANG data model, it is not practical to model the precise data contained in the updates as part of the notification. This is because the specific data nodes supported depend on the implementing system and may even vary dynamically. Therefore, to capture this data, a single parameter that can represent any datastore contents is used, not parameters that represent data nodes one at a time.

A push-change-update notification is the most common type of update for on-change subscriptions. It is not used for periodic subscriptions. The update record in this case contains a data snippet that indicates the full set of changes that data nodes have undergone since the last notification of YANG objects. In other words, this indicates which data nodes have been created, deleted, or have had changes to their values. The format of the data snippet follows YANG-patch [I-D.ietf-netconf-yang-patch], i.e. the same format that would be used with a YANG-patch operation to apply changes to a data tree, indicating the creates, deletes, and modifications of data nodes. Please note that as the update can include a mix of configuration and operational data

The following is an example of push notification. It contains an update for subscription 1011, including a subtree with root foo that contains a leaf, bar:

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-update
       xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>1011</subscription-id>
     <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
     <datastore-contents-xml>
        <foo>
           <bar>some_string</bar>
        </foo>
     </datastore-contents-xml>
   </push-update>
</notification>
```

Figure 1: Push example

The following is an example of an on-change notification. It contains an update for subscription 89, including a new value for a leaf called beta, which is a child of a top-level container called alpha:

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-change-update xmlns=
       "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>89</subscription-id>
     <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
     <datastore-changes-xml>
       <alpha xmlns="http://example.com/sample-data/1.0" >
         <beta>1500</beta>
       </alpha>
     </datastore-changes-xml>
   </push-change-update>
</notification>
```

Figure 2: Push example for on change

The equivalent update when requesting json encoding:

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-change-update xmlns=
       "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>89</subscription-id>
     <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
     <datastore-changes-json>
       {
        "ietf-yang-patch:yang-patch": {
        "patch-id": [
          null
        ],
        "edit": [
          {
              "edit-id": "edit1",
              "operation": "merge",
              "target": "/alpha/beta",
              "value": {
                  "beta": 1500
              }
          }
        ]
       }
     }
     </datastore-changes-json>
   </push-change-update>
</notification>
```

Figure 3: Push example for on change with JSON

When the beta leaf is deleted, the publisher may send

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-change-update xmlns=
       "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>89</subscription-id>
     <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
     <datastore-changes-xml>
       <alpha xmlns="http://example.com/sample-data/1.0" >
         <beta urn:ietf:params:xml:ns:netconf:base:1.0:
             operation="delete"/>
       </alpha>
     </datastore-changes-xml>
   </push-change-update>
</notification>
```

Figure 4: 2nd push example for on change update

3.7. Subscription management

A [[I-D.ietf-netconf-5277bis] subscription needs enhancment to support YANG Push subscription negotiation. Specifically, these enhancements are needed to signal to the subscriber why an attempt has failed.

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, the lack of read authorization on the requested data node, or the inability of the publisher to provide a stream with the requested semantics. In such cases, no subscription is established. Instead, the subscription-result with the failure reason is returned as part of the RPC response. In addition, a set of alternative subscription parameters MAY be returned that would likely have resulted in acceptance of the subscription request, which the subscriber may try for a future subscription attempt.

It should be noted that a rejected subscription does not result in the generation of an rpc-reply with an rpc-error element, as neither the specification of YANG-push specific errors nor the specification of additional data parameters to be returned in an error case are supported as part of a YANG data model.

For instance, for the following request:

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

Figure 5: Establish-Subscription example

the publisher might return:

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

Figure 6: Error response example

3.8. Other considerations

3.8.1. Authorization

A receiver of subscription data may only be sent updates for which they have proper authorization. Data that is being pushed therefore needs to be subjected to a filter that applies all corresponding rules applicable at the time of a specific pushed update, silently removing any non-authorized data from subtrees.

The authorization model for data in YANG datastores is described in the Netconf Access Control Model [RFC6536]. However, some clarifications to that RFC are needed so that the desired access control behavior is applied to pushed updates.

One of these clarifications is that a subscription may only be established if the receiver has read access to the target data node.

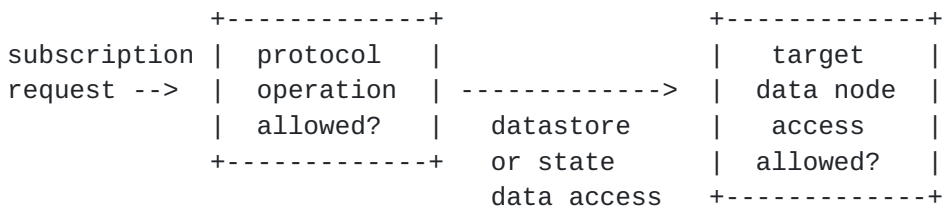

Figure 7: Access control for subscription

Likewise if a receiver no longer has read access permission to a target data node, the subscription must be abnormally terminated (with loss of access permission as the reason provided).

Another clarification to [RFC6536] is that each of the individual nodes in a pushed update must also go through access control filtering. This includes new nodes added since the last update notification, as well as existing nodes. For each of these read access must be verified. The methods of doing this efficiently are left to implementation.

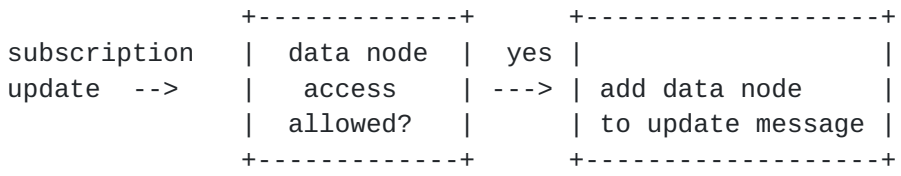

Figure 8: Access control for push updates

If there are read access control changes applied under the target node, no notifications indicating the fact that this has occurred should be provided.

3.8.2. Robustness and reliability considerations

Particularly in the case of on-change push updates, it is important that push updates do not get lost.

Update notifications will typically traverse a secure and reliable transport. Notifications will not be reordered, and will also contain a time stamp. Despite these protections for on-change, it is possible that complete update notifications get lost. For this reason, patch-ids may be included in a subscription so that an application can determine if an update has been lost.

At the same time, it is conceivable that under certain circumstances, a publisher will recognize that it is unable to include within an update notification the full set of objects desired per the terms of

a subscription. In this case, the publisher must take one or more of the following actions.

- A publisher must set the updates-not-sent flag on any update notification which is known to be missing information.
- It may choose to suspend and resume a subscription as per [I-D.ietf-netconf-5277bis].
- When resuming an on-change subscription, the publisher should generate a complete patch from the previous update notification. If this is not possible and the synch-on-start option is configured, then the full datastore contents may be sent instead (effectively replacing the previous contents). If neither of these are possible, then an updates-not-sent flag must be included on the next push-change-update.

3.8.3. Update size and fragmentation considerations

Depending on the subscription, the volume of updates can become quite large. There is no inherent limitation to the amount of data that can be included in a notification. That said, it may not always be practical to send the entire update in a single chunk. Implementations MAY therefore choose, at their discretion, to "chunk" updates and break them out into several update notifications.

3.8.4. Push data streams

There are several conceptual data streams introduced in this specification:

- yang-push includes the entirety of YANG data, including both configuration and operational data.
- operational-push includes all operational (read-only) YANG data
- config-push includes all YANG configuration data.

It is conceivable to introduce other data streams with more limited scope, for example:

- operdata-nocounts-push, a datastream containing all operational (read-only) data with the exception of counters
- other custom datastreams

Those data streams make particular sense for use cases involving service assurance (not relying on operational data), and for use

cases requiring on-change update triggers which make no sense to support in conjunction with fast-changing counters. While it is possible to specify subtree filters on yang-push to the same effect, having those data streams greatly simplifies articulating subscriptions in such scenarios.

3.8.5. Implementation considerations

Implementation specifics are outside the scope of this specification. That said,it should be noted that monitoring of operational state changes inside a system can be associated with significant implementation challenges.

Even periodic retrieval and push of operational counters may consume considerable system resources. In addition the on-change push of small amounts of configuration data may, depending on the implementation, require invocation of APIs, possibly on an object-by-object basis, possibly involving additional internal interrupts, etc.

For those reasons, it is important for an implementation to understand what subscriptions it can or cannot support. It is far preferable to decline a subscription request then to accept such a request when it cannot be met.

Whether or not a subscription can be supported will in general be determined by a combination of several factors, including the subscription policy (on-change or periodic, with on-change in general being the more challenging of the two), the period in which to report changes (1 second periods will consume more resources than 1 hour periods), the amount of data in the subtree that is being subscribed to, and the number and combination of other subscriptions that are concurrently being serviced.

When providing access control to every node in a pushed update, it is possible to make and update efficient access control filters for an update. These filters can be set upon subscription and applied against a stream of updates. These filters need only be updated when (a) there is a new node added/removed from the subscribed tree with different permissions than its parent, or (b) read access permissions have been changed on nodes under the target node for the subscriber.

3.8.6. Not Notifiable YANG Objects

In some cases, a publisher supporting "on-change" notifications may not be able to push updates for some object types "on-change". Reasons for this might be that the value of the data node changes frequently (e.g., a received-octets-counter), that small object changes are frequent and meaningless (e.g., a temperature gauge

changing 0.1 degrees), or that the implementation is not capable of on-change notification of an object type.

The default assumption is that changes on all data nodes will be reported on-change. However if a certain data node cannot do this, it SHOULD be marked with the YANG extension not-notifiable-on-change.

4. A YANG data model for management of datastore push subscriptions

4.1. Overview

The YANG data model for datastore push subscriptions is depicted in the following figure. Following Yang tree convention in the depiction, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parantheses with a name in the middle enclose choice and case nodes. A "+" at the end of a line indicates that the line is to be concatenated with the subsequent line. New YANG tree notation is the i] which indicates that the node in that line has been brought in / imported from another model, and an (a) which indicates this is the specific imported node augmented. In the figure below, all have been imported from 5277bis. The model consists mostly of augmentations to RPCs and notifications defined in the data model for subscriptions for event notifications of [I-D.ietf-netconf-5277bis].

```
module: ietf-yang-push
i]     +--ro streams
i]     |  +--ro stream*   stream
i]     +--rw filters
i]     |  +--rw filter* [filter-id]
i]     |     +--rw filter-id    filter-id
i]     |     +--rw (filter-type)?
i]     |        +--:(rfc5277)
i]     |        |  +--rw filter?
       |        +--:(update-filter)
       |           +--rw (update-filter)?
       |              +--:(subtree)
       |              |  +--rw subtree-filter?
       |              +--:(xpath)
       |                 +--rw xpath-filter? yang:xpath1.0
i]     +--rw subscription-config {configured-subscriptions}?
i]     |  +--rw subscription* [subscription-id]
i]     |     +--rw subscription-id          subscription-id
i]     |     +--rw stream?                  stream
i]     |     +--rw encoding?                encoding
(a)    |     +--rw (filter-type)?
i]     |     |  +--:(rfc5277)
```

```
   i]     |     |  |  +--rw filter?
          |     |  +--:(update-filter)
          |     |  |  +--rw (update-filter)?
          |     |  |     +--:(subtree)
          |     |  |     |  +--ro subtree-filter?
          |     |  |     +--:(xpath)
          |     |  |        +--rw xpath-filter? yang:xpath1.0
   i]     |     |  +--:(by-reference)
   i]     |     |     +--rw filter-ref?         filter-ref
   i]     |     +--rw startTime?                yang:date-and-time
   i]     |     +--rw stopTime?                 yang:date-and-time
          |     +--rw (update-trigger)?
          |     |  +--:(periodic)
          |     |  |  +--rw period              yang:timeticks
          |     |  |  +--rw anchor-time?        yang:date-and-time
          |     |  +--:(on-change) {on-change}?
          |     |     +--rw no-synch-on-start?  empty
          |     |     +--rw dampening-period    yang:timeticks
          |     |     +--rw excluded-change*    change-type
   i]     |     +--rw receivers
   i]     |     |  +--rw receiver* [address]
   i]     |     |     +--rw address             inet:host
   i]     |     |     +--rw port                inet:port-number
   i]     |     |     +--rw protocol?           transport-protocol
   i]     |     +--rw (push-source)?
   i]     |     |  +--:(interface-originated)
   i]     |     |  |  +--rw source-interface?   if:interface-ref
   i]     |     |  +--:(address-originated)
   i]     |     |     +--rw source-vrf?         uint32
   i]     |     |     +--rw source-address    inet:ip-address-no-zone
          |     +--rw dscp?                     inet:dscp
          |     +--rw subscription-priority?    uint8
          |     +--rw subscription-dependency?  string
   (a)    +--ro subscriptions
   i]        +--ro subscription*
   i]           +--ro subscription-id
   i]           +--ro configured-subscription?
   i]           +--ro subscription-status?
   i]           +--ro stream?
   i]           +--ro encoding?
   (a)          +--ro (filter-type)?
   i]           |  +--:(rfc5277)
   i]           |  |  +--ro filter?
                |  +--:(update-filter)
                |  |  +--ro (update-filter)?
                |  |     +--:(subtree)
                |  |     |  +--ro subtree-filter?
                |  |     +--:(xpath)
```

```
                |  |       +--ro xpath-filter?
   i]           |  +--:(by-reference)
   i]           |     +--ro filter-ref?
   i]           +--ro startTime?
   i]           +--ro stopTime?
                +--ro (update-trigger)?
                |  +--:(periodic)
                |  |  +--ro period
                |  |  +--ro anchor-time?
                |  +--:(on-change) {on-change}?
                |     +--ro no-synch-on-start?
                |     +--ro dampening-period
                |     +--ro excluded-change*
   i]           +--ro receivers
   i]           |  +--ro receiver*
   i]           |     +--ro address
   i]           |     +--ro port
   i]           |     +--ro protocol?
   i]           +--ro (push-source)?
   i]           |  +--:(interface-originated)
   i]           |  |  +--ro source-interface?
   i]           |  +--:(address-originated)
   i]           |     +--ro source-vrf?
   i]           |     +--ro source-address
                +--ro dscp?
                +--ro subscription-priority?
                +--ro subscription-dependency?
   i]   rpcs:
   i]     +---x establish-subscription
   (a)    |  +---w input
   i]     |  |  +---w stream?
   i]     |  |  +---w encoding?
   (a)    |  |  +---w (filter-type)?
   i]     |  |  |  +--:(rfc5277)
   i]     |  |  |  |  +---w filter?
          |  |  |  +--:(update-filter)
          |  |  |  |  +---w (update-filter)?
          |  |  |  |     +--:(subtree)
          |  |  |  |     |  +---w subtree-filter?
          |  |  |  |     +--:(xpath)
          |  |  |  |        +---w xpath-filter?
   i]     |  |  |  +--:(by-reference)
   i]     |  |  |     +---w filter-ref?
   i]     |  |  +---w startTime?
   i]     |  |  +---w stopTime?
          |  |  +---w (update-trigger)?
          |  |  |  +--:(periodic)
          |  |  |  |  +---w period
```

```
          |  |  |  |  +---w anchor-time?
          |  |  |  +--:(on-change) {on-change}?
          |  |  |     +---w no-synch-on-start?
          |  |  |     +---w dampening-period
          |  |  |     +---w excluded-change*
          |  |  +---w dscp?
          |  |  +---w subscription-priority?
          |  |  +---w subscription-dependency?
   i]     |  +--ro output
   i]     |     +--ro subscription-result
   i]     |     +--ro (result)?
   i]     |        +--:(success)
   i]     |        |  +--ro subscription-id
   (a)    |        +--:(no-success)
   i]     |           +--ro stream?
   i]     |           +--ro encoding?
   (a)    |           +--ro (filter-type)?
   i]     |           |  +--:(rfc5277)
   i]     |           |  |  +--ro filter?
          |           |  +--:(update-filter)
          |           |  |  +---w (update-filter)?
          |           |  |     +--:(subtree)
          |           |  |     |  +---w subtree-filter?
          |           |  |     +--:(xpath)
          |           |  |        +---w xpath-filter?
   i]     |           |  +--:(by-reference)
   i]     |           |     +--ro filter-ref?
   i]     |           +--ro startTime?
   i]     |           +--ro stopTime?
          |           +--ro (update-trigger)?
          |           |  +--:(periodic)
          |           |  |  +--ro period
          |           |  |  +--ro anchor-time?
          |           |  +--:(on-change) {on-change}?
          |           |     +--ro no-synch-on-start?
          |           |     +--ro dampening-period
          |           |     +--ro excluded-change*
          |           +--ro dscp?
          |           +--ro subscription-priority?
          |           +--ro subscription-dependency?
   i]     +---x modify-subscription
   i]     |  +---w input
   i]     |  |  +---w subscription-id?
   i]     |  |  +---w (filter-type)?
   i]     |  |  |  +--:(rfc5277)
   i]     |  |  |  |  +---w filter?
          |  |  |  +--:(update-filter)
          |  |  |  |  +---w (update-filter)?
```

```
          |  |  |  |     +--:(subtree)
          |  |  |  |     |  +---w subtree-filter?
          |  |  |  |     +--:(xpath)
          |  |  |  |        +---w xpath-filter?
   i]     |  |  |  +--:(by-reference)
   i]     |  |  |     +---w filter-ref?
   i]     |  |  +---w startTime?
   i]     |  |  +---w stopTime?
          |  |  +---w (update-trigger)?
          |  |     +--:(periodic)
          |  |     |  +---w period
          |  |     |  +---w anchor-time?
          |  |     +--:(on-change) {on-change}?
          |  |        +---w dampening-period
          |  |        +---w excluded-change*
   i]     |  +--ro output
   i]     |     +--ro subscription-result
   i]     |     +--ro (result)?
   i]     |        +--:(success)
   i]     |        |  +--ro subscription-id
   i]     |        +--:(no-success)
   i]     |           +--ro stream?
   i]     |           +--ro encoding?
   i]     |           +--ro (filter-type)?
   i]     |           |  +--:(rfc5277)
   i]     |           |  |  +--ro filter?
          |           |  +--:(update-filter)
          |           |  |  +---w (update-filter)?
          |           |  |     +--:(subtree)
          |           |  |     |  +---w subtree-filter?
          |           |  |     +--:(xpath)
          |           |  |        +---w xpath-filter?
   i]     |           |  +--:(by-reference)
   i]     |           |     +--ro filter-ref?
   i]     |           +--ro startTime?
   i]     |           +--ro stopTime?
          |           +--ro (update-trigger)?
          |           |  +--:(periodic)
          |           |  |  +--ro period
          |           |  |  +--ro anchor-time?
          |           |  +--:(on-change) {on-change}?
          |           |     +--ro no-synch-on-start?
          |           |     +--ro dampening-period
          |           |     +--ro excluded-change*
          |           +--ro dscp?
          |           +--ro subscription-priority?
          |           +--ro subscription-dependency?
   i]     +---x delete-subscription
```

```
i]        +---w input
i]        |  +---w subscription-id
i]        +--ro output
i]           +--ro subscription-result

(a)      notifications
(a)    +---n subscription-started
i]     |  +--ro subscription-id
i]     |  +--ro stream?
i]     |  +--ro encoding?
(a)    |  +--ro (filter-type)?
i]     |  |  +--:(rfc5277)
i]     |  |  |  +--ro filter?
       |  |  +--:(update-filter)
       |  |  |     +--rw (update-filter)?
       |  |  |        +--:(subtree)
       |  |  |        |  +--ro subtree-filter?
       |  |  |        +--:(xpath)
       |  |  |           +--rw xpath-filter?
i]     |  |  +--:(by-reference)
i]     |  |     +--ro filter-ref?
i]     |  +--ro startTime?
i]     |  +--ro stopTime?
       |  +--ro (update-trigger)?
       |  |      +--:(periodic)
       |  |  |  +--ro period
       |  |  |  +--ro anchor-time?
       |  |  +--:(on-change) {on-change}?
       |  |     +--ro no-synch-on-start?
       |  |     +--ro dampening-period
       |  |     +--ro excluded-change*
       |  +--ro dscp?
       |  +--ro subscription-priority?
       |  +--ro subscription-dependency?
(a)    +---n subscription-modified
i]     |  +--ro subscription-id
i]     |  +--ro stream?
i]     |  +--ro encoding?
(a)    |  +--ro (filter-type)?
i]     |  |  +--:(rfc5277)
i]     |  |  |  +--ro filter?
       |  |  +--:(update-filter)
       |  |  |     +--rw (update-filter)?
       |  |  |        +--:(subtree)
       |  |  |        |  +--ro subtree-filter?
       |  |  |        +--:(xpath)
       |  |  |           +--rw xpath-filter?
i]     |  |  +--:(by-reference)
```

```
i]     |  |     +--ro filter-ref?
i]     |  +--ro startTime?
i]     |  +--ro stopTime?
       |  +--ro (update-trigger)?
       |  |      +--:(periodic)
       |  |  |  +--ro period
       |  |  |  +--ro anchor-time?
       |  |  +--:(on-change) {on-change}?
       |  |     +--ro no-synch-on-start?
       |  |     +--ro dampening-period
       |  |     +--ro excluded-change*
       |  +--ro dscp?
       |  +--ro subscription-priority?
       |  +--ro subscription-dependency?
i]     +---n subscription-terminated
i]     |  +--ro subscription-id
i]     |  +--ro reason?
i]     +---n subscription-suspended
i]     |  +--ro subscription-id
i]     |  +--ro reason?
i]     +---n subscription-resumed
i]     |  +--ro subscription-id
i]     +---n replay-complete
i]     |  +--ro subscription-id
i]     +---n notification-complete
i]     |  +--ro subscription-id
       +---n push-update
       |  +--ro subscription-id
       |  +--ro time-of-update?
       |  +--ro updates-not-sent?
       |  +--ro datastore-contents?
       +---n push-change-update {on-change}?
          +--ro subscription-id
          +--ro time-of-update?
          +--ro updates-not-sent?
          +--ro datastore-changes?
```

Figure 9: Model structure

The components of the model are described in the following subsections.

4.2. Update streams

Container "update-streams" is used to indicate which data streams are provided by the system and can be subscribed to. For this purpose, it contains a leaf list of data nodes identifying the supported streams.

4.3. Filters

Container "filters" contains a list of configurable data filters, each specified in its own list element. This allows users to configure filters separately from an actual subscription, which can then be referenced from a subscription. This facilitates the reuse of filter definitions, which can be important in case of complex filter conditions.

One of three types of filters can be specified as part of a filter list element. Subtree filters follow syntax and semantics of RFC 6241 and allow to specify which subtree(s) to subscribe to. In addition, XPath filters can be specified for more complex filter conditions. Finally, filters can be specified using syntax and semantics of RFC5277.

It is conceivable to introduce other types of filters; in that case, the data model needs to be augmented accordingly.

4.4. Subscription configuration

As an optional feature, configured-subscriptions, allows for the configuration of subscriptions as opposed to RPC. Subscriptions configurations are represented by list subscription-config. Each subscription is represented through its own list element and includes the following components:

- "subscription-id" is an identifier used to refer to the subscription.

- "stream" refers to the stream being subscribed to. The subscription model assumes the presence of perpetual and continuous streams of updates. Various streams are defined: "push-update" covers the entire set of YANG data in the publisher. "operational-push" covers all operational data, while "config-push" covers all configuration data. Other streams could be introduced in augmentations to the model by introducing additional identities.

- "encoding" refers to the encoding requested for the data updates. By default, updates are encoded using XML. However, JSON can be requested as an option if the json-enconding feature is supported. Other encodings may be supported in the future.

- "anchor-time" is a timestamp. When used in conjunction with period, the boundaries of periodic update periods may be calculated.

- Filters for a subscription can be specified using a choice, allowing to either reference a filter that has been separately configured or entering its definition inline.

- A choice of subscription policies allows to define when to send new updates - periodic or on change.

  * For periodic subscriptions, the trigger is defined by a "period", a parameter that defines the interval with which updates are to be pushed. The start time of the subscription serves as anchor time, defining one specific point in time at which an update needs to be sent. Update intervals always fall on the points in time that are a multiple of a period after the start time.

  * For on-change subscriptions, the trigger occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that is guided by additional parameters. "dampening-period" specifies the interval that must pass before a successive update for the same data node is sent. The first time a change is detected, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed, containing the value of the data node that is then valid. "excluded-change" allows to restrict the types of changes for which updates are sent (changes to object values, object creation or deletion events). "no-synch-on-start" is a flag that allows to specify whether or not a complete update with all the subscribed data should be sent at the beginning of a subscription; if the flag is omitted, a complete update is sent to facilitate synchronization. It is conceivable to augment the data model with additional parameters in the future to specify even more refined policies, such as parameters that specify the magnitude of a change that must occur before an update is triggered.

- This is followed with a list of receivers for the subscription, indicating for each receiver the transport that should be used for push updates (if options other than Netconf are supported). It should be noted that the receiver does not have to be the same system that configures the subscription.

- Finally, "push-source" can be used to specify the source of push updates, either a specific interface or publisher address.

A subscription established through configuration cannot be deleted using an RPC. Likewise, subscriptions established through RPC cannot be deleted through configuration.

The deletion of a subscription, whether through RPC or configuration, results in immediate termination of the subsciption.

4.5. Subscription monitoring

Subscriptions can be subjected to management themselves. For example, it is possible that a publisher may no longer be able to serve a subscription that it had previously accepted. Perhaps it has run out of resources, or internal errors may have occurred. When this is the case, a publisher needs to be able to temporarily suspend the subscription, or even to terminate it. More generally, the publisher should provide a means by which the status of subscriptions can be monitored.

Container "subscriptions" contains the state of all subscriptions that are currently active. This includes subscriptions that were established (and have not yet been deleted) using RPCs, as well as subscriptions that have been configured as part of configuration.

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

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

4.6. Notifications

4.6.1. Monitoring and OAM Notifications

OAM notifications are reused from [I-D.ietf-netconf-5277bis]. Some have augmentations to include new objects defined in this draft.

Still to be investigated is whether a publisher might also provide additional information about subscriptions, such as statistics about the number of data updates that were sent. However, such information is currently outside the scope of this specification.

4.6.2. Update Notifications

The data model introduces two YANG notifications for the actual updates themselves.

Notification "push-update" is used to send a complete snapshot of the data that has been subscribed to, with all YANG object filters applied. The notification is used for periodic subscription updates in a periodic subscription.

The notification can also be used in an on-change subscription for the purposes of allowing a receiver to "synch". Specifically, it is used at the start of an on-change subscription, unless no-synch-on-start is specified for the subscription. In addition, it MAY be used during the subscription, for example if change updates were not sent as indicated by the "updates-not-sent" flag (see below), or for synch updates at longer period intervals (such as once per day) to mitigate the possibility of any application-dependent synchronization drift. The trigger for sending a push-update notification in conjunction with on-change subscriptions are at this point outside the scope of the specification.

The format and syntax of the contained data corresponds to the format and syntax of data that would be returned in a corresponding get operation with the same filter parameters applied.

Notification "push-change-update" is used to send data updates for changes that have occurred in the subscribed data. This notification is used only in conjunction with on-change subscriptions.

The data updates are encoded analogous to the syntax of a corresponding yang-patch operation. It corresponds to the data that would be contained in a yang-patch operation applied to the YANG datastore at the previous update, to result in the current state (and applying it also to operational data).

In rare circumstances, the notification can include a flag "updates-not-sent". This is a flag which indicates that not all changes which have occurred since the last update are actually included with this update. In other words, the publisher has failed to fulfill its full subscription obligations, for example in cases where it was not able to keep up with a change burst. To facilitate synchronization, a publisher MAY subsequently send a push-update containing a full snapshot of subscribed data. Such a push-update might also be triggered by a subscriber requesting an on-demand synchronization.

4.7. RPCs

YANG-Push subscriptions are established, modified, and deleted using three RPCs.

4.7.1. Establish-subscription RPC

The subscriber sends an establish-subscription RPC with the parameters in section 3.1. For instance

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

Figure 10: Establish-subscription RPC

The publisher must respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the subscription-id of the accepted subscription. In that case a publisher may respond:

```
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <subscription-result
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       ok
    </subscription-result>
    <subscription-id
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       52
    </subscription-id>
</rpc-reply>
```

Figure 11: Establish-subscription positive RPC response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, the lack of read authorization on the requested data node, or the inability of the publisher to provide a stream with the requested semantics.

When the requester is not authorized to read the requested data node, the returned "error-info"; indicates an authorization error and the requested node. For instance, if the above request was unauthorized to read node "ex:foo" the publisher may return:

```
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <subscription-result
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       error-data-not-authorized
    </subscription-result>
</rpc-reply>
```

Figure 12: Establish-subscription access denied response

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error what subscription parameters would have been accepted for the request. However, there are no guarantee that subsequent requests for this subscriber or others will in fact be accepted.

For example, for the following request:

```
<netconf:rpc message-id="101"
   xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
   <establish-subscription
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      <stream>push-update</stream>
      <filter netconf:type="xpath"
            xmlns:ex="http://example.com/sample-data/1.0"
            select="/ex:foo"/>
      <dampening-period>10</dampening-period>
      <encoding>encode-xml</encoding>
   </establish-subscription>
</netconf:rpc>
```

Figure 13: Establish-subscription request example 2

A publisher that cannot serve on-change updates but periodic updates might return the following:

```
<rpc-reply message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <subscription-result
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       error-no-such-option
    </subscription-result>
    <period>100</period>
</rpc-reply>
```

Figure 14: Establish-subscription error response example 2

4.7.2. Modify-subscription RPC

The subscriber may send a modify-subscription PRC for a subscription previously established using RPC The subscriber may change any subscription parameters by including the new values in the modify-subscription RPC. Parameters not included in the rpc should remain unmodified. For illustration purposes we include an exchange example where a subscriber modifies the period of the subscription.

```
<netconf:rpc message-id="102"
   xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
   <modify-subscription
        xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      <stream>push-update</stream>
      <subscription-id>
         1011
      </subscription-id>
      <filter netconf:type="xpath"
           xmlns:ex="http://example.com/sample-data/1.0"
           select="/ex:foo"/>
      <period>250</period>
      <encoding>encode-xml</encoding>
   </modify-subscription>
</netconf:rpc>
```

Figure 15: Modify subscription request

The publisher must respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the subscription-id of the accepted subscription. In that case a publisher may respond:

```
<rpc-reply message-id="102"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <subscription-result
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      ok
   </subscription-result>
   <subscription-id
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      1011
   </subscription-id>
</rpc-reply>
```

Figure 16: Modify subscription response

If the subscription modification is rejected, the publisher must send a response like it does for an establish-subscription and maintain the subscription as it was before the modification request. A subscription may be modified multiple times.

A configured subscription cannot be modified using modify-subscription RPC. Instead, the configuration needs to be edited as needed.

4.7.3. Delete-subscription RPC

To stop receiving updates from a subscription and effectively delete a subscription that had previously been established using an establish-subscription RPC, a subscriber can send a delete-subscription RPC, which takes as only input the subscription-id. For example:

```
<netconf:rpc message-id="103"
   xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
   <delete-subscription
         xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      <subscription-id>
         1011
      </subscription-id>
   </delete-subscription>
</netconf:rpc>

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

Figure 17: Delete subscription

Configured subscriptions cannot be deleted via RPC, but have to be removed from the configuration.

4.7.4. YANG Module Synchronization

In order to fully support datastore replication, the receiver needs to know the YANG module library that is in use by server that is being replicated. The YANG 1.0 module library information is sent by a NETCONF server in the NETCONF 'hello' message. For YANG 1.1 modules and all modules used with the RESTCONF [I-D.ietf-netconf-restconf] protocol, this information is provided by the YANG Library module (ietf-yang-library.yang from [RFC7895]. The YANG library information is important for the receiver to reproduce the set of object definitions used by the replicated datastore.

The YANG library includes a module list with the name, revision, enabled features, and applied deviations for each YANG YANG module implemented by the server. The receiver is expected to know the YANG library information before starting a subscription. The "/modules-state/module-set-id" leaf in the "ietf-yang-library" module can be used to cache the YANG library information. [ED. NOTE: Should "module-set-id" be added to establish-subscription response?]

The set of modules, revisions, features, and deviations can change at run-time (if supported by the server implementation). In this case, the receiver needs to be informed of module changes before data nodes from changed modules can be processed correctly. The YANG library provides a simple "yang-library-change" notification that informs the client that the library has changed somehow. The receiver then needs to re-read the entire YANG library data for the replicated server in order to detect the specific YANG library changes. The "ietf-netconf-notifications" module defined in [RFC6470] contains a "netconf-capability-change" notification that can identify specific module changes. For example, the module URI capability of a newly loaded module will be listed in the "added-capability" leaf-list, and the module URI capability of an removed module will be listed in the "deleted-capability" leaf-list.

5. YANG module

```
<CODE BEGINS> file "ietf-yang-push@2016-10-28.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

  import ietf-inet-types {
    prefix inet;
```

```
    }
    import ietf-yang-types {
      prefix yang;
    }
    import ietf-event-notifications {
      prefix notif-bis;
    }

    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:   Alexander Clemm
                 <mailto:alex@sympotech.com>

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

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

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

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

       Editor:   Andy Bierman
                 <mailto:andy@yumaworks.com>

       Editor:   Balazs Lengyel
                 <mailto:balazs.lengyel@ericsson.com>";

    description
      "This module contains conceptual YANG specifications
       for YANG push.";

    revision 2016-10-28 {
      description
        "Updates to simplify modify-subscription, add anchor-time";
      reference "YANG Datastore Push, draft-ietf-netconf-yang-push-04";
```

```
    }

    feature on-change {
      description
        "This feature indicates that on-change updates are
         supported.";
    }

   /*
    * IDENTITIES
    */

    /* Additional errors for subscription operations */
    identity error-data-not-authorized {
      base notif-bis:error;
      description
        "No read authorization for a requested data node.";
    }

    /* Additional types of streams */
    identity update-stream {
      description
        "Base identity to represent a conceptual system-provided
         datastream of datastore updates with predefined semantics.";
    }

    identity yang-push {
      base update-stream;
      description
        "A conceptual datastream consisting of all datastore
         updates, including operational and configuration data.";
    }

    identity operational-push {
      base update-stream;
      description
        "A conceptual datastream consisting of updates of all
         operational data.";
    }

    identity config-push {
      base update-stream;
      description
        "A conceptual datastream consisting of updates of all
         configuration data.";
    }
```

```
    identity custom-stream {
      base update-stream;
      description
        "A conceptual datastream for datastore
         updates with custom updates as defined by a user.";
    }

    /* Additional transport option */
    identity http2 {
      base notif-bis:transport;
      description
        "HTTP2 notifications as a transport";
    }

    /*
     * TYPE DEFINITIONS
     */

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

    typedef change-type {
      type enumeration {
        enum "create" {
          description
            "A new data node was created";
        }
        enum "delete" {
          description
            "A data node was deleted";
        }
        enum "modify" {
          description
            "The value of a data node has changed";
        }
      }
      description
        "Specifies different types of changes that may occur
         to a datastore.";
    }

    typedef update-stream {
      type identityref {
        base update-stream;
```

```
      }
      description
        "Specifies a system-provided datastream.";
    }

    grouping update-filter {
      description
        "This groupings defines filters for push updates for a
        datastore tree.  The filters define which updates are of
        interest in a push update subscription.  Mixing and matching
        of multiple filters does not occur at the level of this
        grouping. When a push-update subscription is created, the
        filter can be a regular subscription filter, or one of the
        additional filters that are defined in this grouping.";
      choice update-filter {
        description
          "Define filters regarding which data nodes to include
           in push updates";
        case subtree {
          description
            "Subtree filter.";
          anyxml subtree-filter {
            description
              "Subtree-filter used to specify the data nodes targeted
              for subscription within a subtree, or subtrees, of a
              conceptual YANG datastore.  Objects matching the filter
              criteria will traverse the filter. The syntax follows
              the subtree filter syntax specified in RFC 6241,
              section 6.";
            reference "RFC 6241 section 6";
          }
        }
        case xpath {
          description
            "XPath filter";
          leaf xpath-filter {
            type yang:xpath1.0;
            description
              "Xpath defining the data items of interest.";
          }
        }
      }
    }

    grouping update-policy {
      description
        "This grouping describes the conditions under which an
         update will be sent as part of an update stream.";
```

```
      choice update-trigger {
        description
          "Defines necessary conditions for sending an event to
           the subscriber.";
        case periodic {
          description
            "The agent is requested to notify periodically the
             current values of the datastore or the subset
             defined by the filter.";
          leaf period {
            type yang:timeticks;
            mandatory true;
            description
              "Duration of time which should occur between periodic
               push updates.  Where the anchor of a start-time is
               available, the push will include the objects and their
               values which exist at an exact multiple of timeticks
               aligning to this start-time anchor.";
          }
          leaf anchor-time {
            type yang:date-and-time;
            description
              "Designates a timestamp from which the series of
              periodic push updates are computed. The next update
              will take place at the next period interval from the
              anchor time.  For example, for an anchor time at the
              top of a minute and a period interval of a minute,
              the next update will be sent at the top of the next
              minute.";
          }
        }
        case on-change {
          if-feature "on-change";
          description
            "The agent is requested to notify changes in
             values in the datastore or a subset of it defined
             by a filter.";
          leaf no-synch-on-start {
            type empty;
            description
              "This leaf acts as a flag that determines behavior at the
               start of the subscription.  When present,
               synchronization of state at the beginning of the
               subscription is outside the scope of the subscription.
               Only updates about changes that are observed from the
               start time, i.e. only push-change-update notifications
               are sent.
               When absent (default behavior), in order to facilitate
```

```
              a receiver's synchronization, a full update is sent
              when the subscription starts using a push-update
              notification, just like in the case of a periodic
              subscription.  After that, push-change-update
              notifications only are sent unless the Publisher chooses
              to resynch the subscription again.";
          }
          leaf dampening-period {
            type yang:timeticks;
            mandatory true;
            description
              "Minimum amount of time that needs to have
               passed since the last time an update was
               provided.";
          }
          leaf-list excluded-change {
            type change-type;
            description
              "Use to restrict which changes trigger an update.
               For example, if modify is excluded, only creation and
               deletion of objects is reported.";
          }
        }
      }
    }

    grouping subscription-qos {
      description
        "This grouping describes Quality of Service information
         concerning a subscription.  This information is passed to lower
         layers for transport priortization and treatment";
      leaf dscp {
        if-feature "notif-bis:configured-subscriptions";
        type inet:dscp;
        default "0";
        description
          "The push update's IP packet transport priority.
           This is made visible across network hops to receiver.
           The transport priority is shared for all receivers of
           a given subscription.";
      }
      leaf subscription-priority {
        type uint8;
        description
          "Relative priority for a subscription.   Allows an
           underlying transport layer perform informed load
           balance allocations between various subscriptions";
      }
```

```
      leaf subscription-dependency {
        type string;
        description
          "Provides the Subscription ID of a parent subscription
           without which this subscription should not exist. In
           other words, there is no reason to stream these objects
           if another subscription is missing.";
      }
    }

    augment "/notif-bis:establish-subscription/notif-bis:input" {
      description
        "Define additional subscription parameters that apply
         specifically to push updates";
      uses update-policy;
      uses subscription-qos;
    }
    augment "/notif-bis:establish-subscription/notif-bis:input/"+
            "notif-bis:filter-type" {
      description
        "Add push filters to selection of filter types.";
      case update-filter {
        description
          "Additional filter options for push subscription.";
        uses update-filter;
      }
    }
    augment "/notif-bis:establish-subscription/notif-bis:output" {
      description
        "Allow to return additional subscription parameters that apply
         specifically to push updates.";
      uses update-policy;
      uses subscription-qos;
    }
    augment "/notif-bis:establish-subscription/notif-bis:output/"+
      "notif-bis:result/notif-bis:no-success/notif-bis:filter-type" {
      description
        "Add push filters to selection of filter types.";
      case update-filter {
        description
          "Additional filter options for push subscription.";
        uses update-filter;
      }
    }
    augment "/notif-bis:modify-subscription/notif-bis:input" {
      description
        "Define additional subscription parameters that apply
         specifically to push updates.";
```

```
      uses update-policy;
    }
    augment "/notif-bis:modify-subscription/notif-bis:input/"+
            "notif-bis:filter-type" {
      description
        "Add push filters to selection of filter types.";
      case update-filter {
        description
          "Additional filter options for push subscription.";
        uses update-filter;
      }
    }
    augment "/notif-bis:modify-subscription/notif-bis:output" {
      description
        "Allow to retun additional subscription parameters that apply
         specifically to push updates.";
      uses update-policy;
      uses subscription-qos;
    }
    augment "/notif-bis:modify-subscription/notif-bis:output/"+
        "notif-bis:result/notif-bis:no-success/notif-bis:filter-type" {
      description
        "Add push filters to selection of filter types.";
      case update-filter {
        description
          "Additional filter options for push subscription.";
        uses update-filter;
      }
    }
    notification push-update {
      description
        "This notification contains a push update, containing
         data subscribed to via a subscription.
         This notification is sent for periodic updates, for a
         periodic subscription.  It can also be used for
         synchronization updates of an on-change subscription.
         This notification shall only be sent to receivers
         of a subscription; it does not constitute a general-purpose
         notification.";
      leaf subscription-id {
        type notif-bis:subscription-id;
        mandatory true;
        description
          "This references the subscription because of which the
           notification is sent.";
      }
      leaf time-of-update {
        type yang:date-and-time;
```

```
        description
          "This leaf contains the time of the update.";
      }
      leaf updates-not-sent {
        type empty;
        description
          "This is a flag which indicates that not all data nodes
           subscribed to are included included with this
           update.  In other words, the publisher has failed to
           fulfill its full subscription obligations.
           This may lead to intermittent loss of synchronization
           of data at the client.  Synchronization at the client
           can occur when the next push-update is received.";
      }
      anydata datastore-contents {
        description
          "This contains the updated data.  It constitutes a snapshot
           at the time-of-update of the set of data that has been
           subscribed to.  The format and syntax of the data
           corresponds to the format and syntax of data that would be
           returned in a corresponding get operation with the same
           filter parameters applied.";
      }
    }
    notification push-change-update {
      if-feature "on-change";
      description
        "This notification contains an on-change push update.
         This notification shall only be sent to the receivers
         of a subscription; it does not constitute a general-purpose
         notification.";
      leaf subscription-id {
        type notif-bis:subscription-id;
        mandatory true;
        description
          "This references the subscription because of which the
           notification is sent.";
      }
      leaf time-of-update {
        type yang:date-and-time;
        description
          "This leaf contains the time of the update, i.e. the
           time at which the change was observed.";
      }
      leaf updates-not-sent {
        type empty;
        description
          "This is a flag which indicates that not all changes which
```

```
          have occured since the last update are included with this
          update.  In other words, the publisher has failed to
          fulfill its full subscription obligations, for example in
          cases where it was not able to keep up with a change burst.
          To facilitate synchronization, a publisher MAY subsequently
          send a push-update containing a full snapshot of subscribed
          data. Such a push-update might also be triggered by a
          subscriber requesting an on-demand synchronization.";
      }
      anydata datastore-changes {
        description
          "This contains datastore contents that has changed
           since the previous update, per the terms of the
           subscription.  Changes are encoded analogous to
           the syntax of a corresponding yang-patch operation,
           i.e. a yang-patch operation applied to the YANG datastore
           implied by the previous update to result in the current
           state (and assuming yang-patch could also be applied to
           operational data).";
      }
    }
    augment "/notif-bis:subscription-started" {
      description
        "This augmentation adds push subscription parameters
         to the notification that a subscription has
         started and data updates are beginning to be sent.
         This notification shall only be sent to receivers
         of a subscription; it does not constitute a general-purpose
         notification.";
      uses update-policy;
      uses subscription-qos;
    }
    augment "/notif-bis:subscription-started/notif-bis:filter-type" {
      description
        "This augmentation allows to include additional update filters
         options to be included as part of the notification that a
         subscription has started.";
      case update-filter {
        description
          "Additional filter options for push subscription.";
        uses update-filter;
      }
    }
    augment "/notif-bis:subscription-modified" {
      description
        "This augmentation adds push subscription parameters
         to the notification that a subscription has
         been modified.
```

```
         This notification shall only be sent to receivers
         of a subscription; it does not constitute a general-purpose
         notification.";
       uses update-policy;
       uses subscription-qos;
     }
     augment "/notif-bis:subscription-modified/notif-bis:filter-type" {
       description
         "This augmentation allows to include additional update
          filters options to be included as part of the notification
          that a subscription has been modified.";
       case update-filter {
         description
           "Additional filter options for push subscription.";
         uses update-filter;
       }
     }
     augment "/notif-bis:filters/notif-bis:filter/"+
             "notif-bis:filter-type" {
       description
         "This container adds additional update filter options
          to the list of configurable filters
          that can be applied to subscriptions.  This facilitates
          the reuse of complex filters once defined.";
       case update-filter {
         uses update-filter;
       }
     }
     augment "/notif-bis:subscription-config/notif-bis:subscription" {
       description
         "Contains the list of subscriptions that are configured,
          as opposed to established via RPC or other means.";
       uses update-policy;
       uses subscription-qos;
     }
     augment "/notif-bis:subscription-config/notif-bis:subscription/"+
             "notif-bis:filter-type" {
       description
         "Add push filters to selection of filter types.";
       case update-filter {
         uses update-filter;
       }
     }
     augment "/notif-bis:subscriptions/notif-bis:subscription" {
       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. establish-subscription, delete-subscription,
        and modify-subscription, as well as subscriptions that
        have been established via configuration.";
      uses update-policy;
      uses subscription-qos;
    }
    augment "/notif-bis:subscriptions/notif-bis:subscription/"+
            "notif-bis:filter-type" {
      description
        "Add push filters to selection of filter types.";
      case update-filter {
        description
          "Additional filter options for push subscription.";
        uses update-filter;
      }
    }
  }
```

<CODE ENDS>

6. Security Considerations

Subscriptions could be used to attempt to overload publishers of YANG datastores. For this reason, it is important that the publisher has the ability to decline a subscription request if it would deplete its resources. In addition, a publisher needs to be able to suspend an existing subscription when needed. When this occur, the subscription status is updated accordingly and the receivers are notified. Likewise, requests for subscriptions need to be properly authorized.

A subscription could be used to retrieve data in subtrees that a receiver has no authorized access to. Therefore it is important that data pushed based on subscriptions is authorized in the same way that regular data retrieval operations are. Data being pushed to a receiver needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

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. Receivers which do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, the Netconf Authorization Control Model SHOULD be used to control and restrict authorization of subscription configuration.

7. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Tim Jenkins, Kent Watsen, Susan Hares, Yang Geng, Peipei Guo, Michael Scharf, Sharon Chisholm, and Guangying Zheng.

8. References

8.1. Normative References

[I-D.ietf-netconf-5277bis]
Clemm, A., Gonzalez Prieto, A., Voit, E., Tripathy, A., Nilsen-Nygaard, E., Chisholm, S., and H. Trevino, "Subscribing to YANG-Defined Event Notifications", draft-ietf-netconf-5277bis-01 (work in progress), October 2016.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-12 (work in progress), September 2016.

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, February 2012.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, June 2016.

[RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, August 2016.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, August 2016.

8.2. Informative References

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-17, September 2016.

[RFC1157] Case, J., "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, June 2016.

Appendix A. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

A.1. Unresolved and yet-to-be addressed issues

Which stream types to introduce, if any based on implications of opstate. Current list includes streams for all operational and for all config data. Consider adding stream for operational data minus counters.

We need a new Metadata filter. But so does traditional GET. This should be relevant independent of subscriptions. This has implications of ephemeral requirements from I2RS

Should we allow an interplay of filter types in a single subscription. Or should we keep them fully independent.

Do we add a counter for the number of object changes during a dampening period?

A.2. Agreement in principal

Do we need an extension for NACM to support filter out datastore nodes for which the receiver has no read access? (And how does this differ from existing GET, which must do the same filtering?) In 5277, such filtering is done at the notification level. Yang-push includes notification-content filtering. This may be very expensive in terms of processing. Andy suggestion: only accept Yang-push subscriptions for subtrees the user has rights for all the nodes in the subtree. Changes to those rights trigger a subscription termination. Should we codify this, or let vendors determine when per subtree filtering might be applied?

Need to add a new RPC to request enabling a resynch for an existing on-change subscription exposed on publisher

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v03 to v04

- Updates-not-sent flag added
- Not notifiable extension added
- Dampening period is for whole subscription, not single objects
- Moved start/stop into rfc5277bis
- Client and Server changed to subscriber, publisher, and receiver
- Anchor time for periodic
- Message format for synchronization (i.e. synch-on-start)
- Material moved into 5277bis
- QoS parameters supported, by not allowed to be modified by RPC
- Text updates throughout

Authors' Addresses

Alexander Clemm
Sympotech

Email: alex@sympotech.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Balazs Lengyel
Ericsson

Email: balazs.lengyel@ericsson.com