NETCONF
Internet-Draft
Intended status: Standards Track
Expires: September 1, 2017

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

February 28, 2017

Subscribing to YANG datastore push updates
draft-ietf-netconf-yang-push-05

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 September 1, 2017.

Copyright Notice

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 WG's Subscribed Notifications draft [I-D:netconf-sub-notif]. At its core, the solution defined here suppliments that work by introducing datastore push update mechanisms, and providing corresponding extensions to the event subscription model. The document also includes YANG data model augmentations which extend the model and RPCs defined within [I-D:netconf-sub-notif].

Key capabilities worth highlighting include:

- Additions to event subscription mechanisms which allow clients to subscribe to 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 filter 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 hints at subscription parameters which might have succeeded.

- 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:netconf-sub-notif]. 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:netconf-sub-notif] and introduces new capabilities that allow subscribers to specify what to include in an update notification and what triggers such an update notification.

- Enhancements to filters. Specifically the filter must at least identify at least one targeted yang data node/subtree. The filter may also define additional yang nodes/subtrees to include or exclude. The publisher must only send to the receiver those data node updates that can traverse applied filter. 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.

- o 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.
  - * 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 specifiing the dampening period. This period is the interval which 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 allowing the restriction of the types of changes for which updates are sent (changes to object values, object creation or deletion events).
    - + A third parameter specifing whether or not a complete push-update with all the subscribed data should be sent at the beginning of a subscription. Such a push provides the receiver the current state, and establish the frame of reference for subsequent updates.
- o Anydata encoding for the contents of periodic and on-change push updates.

The subscription data model is described via augmentations to [I-D:netconf-sub-notif] later in this specification. It is conceivable that additional subscription parameters might be interesting. Augmentations to the subscription data model may be used for this.

3.2. Negotiation of Subscription Policies

A dynamic subscription request SHOULD be declined based on publisher's assessment that it may be unable to provide a filtered update notifications that would meet the terms of the request. But a

subscriber may quickly follow up with a new subscription request using different parameters.

Random guessing at different parameters should be discouraged. Therefore to minimize the number of subscription iterations between subscriber and publisher, dynamic subscriptions must support a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is limited to either an establish or modify subscription request, followed by no-success response. The no-success message, where available SHOULD include in the returned error information parameters. The returned parameters provide information 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.

Negotiable parameters which may be returned from a publisher beyond those from [I-D:netconf-sub-notif] include: hints at acceptable time intervals, size estimates forn the number or objects which would be returned from a filter, and the names of targeted objects not found in the publisher’s YANG tree.

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, only the objects supporting on-change will be provided. For more on how objects are so marked, see Section 3.8.5

Any updates for an on-change subscription will include only supported objects for which a change was detected and which met the filtering criteria. 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 subscription are sent until the end of the dampening

period. Values sent at the end of the dampening period are the current values of all changed objects which are current at the time the dampening period expires. Changed objects includes those which were deleted or newly created during that dampening period.

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.

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 [RFC8072]. 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 or 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 push-update 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. To capture this data, a single parameter that can encapsulate the full set of subscribed 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. 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 [RFC8072], 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:netconf-sub-notif] 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 every data node specifically named within the subscription filter.

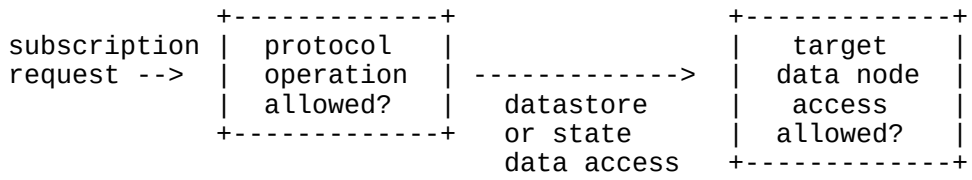

Figure 7: Access control for subscription

Likewise if a receiver no longer has read access permission to a data node named/targeted within a filter, 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.

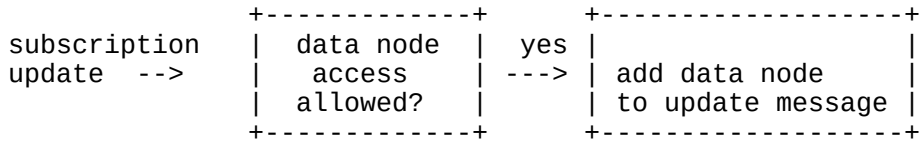

Figure 8: Access control for push updates

If there are read access control changes applied under the data node named/targeted within a filter, 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, update sequence numbers for push-change-updates 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:netconf-sub-notif].
- 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. Additionally, based on the platform, it is possible that push-updates for a single subscription are best sent independently from different line-cards. Therefore, it may not always be practical to send the entire update in a single chunk. Implementations of push-update MAY therefore choose, at their discretion, to "chunk" updates and break them out into several push-update notifications. In this case the updates-not-sent flag will indicate that no single push-update is complete. Push-change-updates may also be chunked as long as none of the changed objects in the separate pushes are state-entangled.

3.8.4. 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.

3.8.5. Identifying on-change 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.

Support for on-change notification is usually specific to the individual YANG model and/or implementation so it is possible to define in design time. System integrators need this information (without reading any data from a live node).

The default assumption is that no data nodes support on-change notification. Schema nodes and subtrees that support on-change notifications MUST be marked as such with the YANG extension notifiable-on-change.

If the model designer wants to add the notifiable-on-change statement to an existing module, but wants to avoid modifying the text of the existing module, the notifiable-on-change statement may be added using deviation statements.

```
 extension notifiable-on-change {
   Indicates whether changes to the data node are reportable in
   on-change subscriptions.

   The statement MUST only be a substatement of the leaf, leaf-list,
   container, list, anyxml, anydata  statements. Zero or One
   notifiable-on-change statement is allowed per parent statement. NO
   substatements are allowed.

   The argument is a boolean value indicating whether on-change
   notifications are supported. If notifiable-on-change is not
   specified, the default is the same as the parent data node's
   value. For top level data nodes the default value is false.";

   argument value;
 }
```

Figure 9: Notifiable Extension

When an on-change subscription is established data-nodes marked with notifiable-on-change false; will be automatically filtered out. This also means that authorization checks need be performed on them.

```
 deviation /sys:system/sys:system-time {
   deviate add {
      yp:notifiable-on-change false;
   }
 }
```

Figure 10: Deviation Example

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:netconf-sub-notif].

(Note: the yp indicates augmentations from yang push above and beyond the event-notifications model)

```
module: ietf-subscribed-notifications
    +--ro streams
    |  +--ro stream*   stream
    +--rw filters
    |  +--rw filter* [identifier]
    |     +--rw identifier           filter-id
    |     +--rw (filter-type)?
    |        +--:(by-reference)
    |        |  +--rw filter-ref?          filter-ref
    |        +--:(event-filter)
    |        |  +--rw filter?
    |        +--:(yp:update-filter)
    |           +--rw (yp:update-filter)?
    |              +--:(yp:subtree)
    |              |  +--rw yp:subtree-filter?
    |              +--:(yp:xpath)
    |                 +--rw yp:xpath-filter?     yang:xpath1.0
    +--rw subscription-config {configured-subscriptions}?
    |  +--rw subscription* [identifier]
    |     +--rw identifier              subscription-id
    |     +--rw stream?                 stream
    |     +--rw encoding?               encoding
    |     +--rw stop-time?              yang:date-and-time
    |     +--rw (filter-type)?
    |     |  +--:(by-reference)
    |     |  |  +--rw filter-ref?             filter-ref
    |     |  +--:(event-filter)
    |     |  |  +--rw filter?
    |     |  +--:(yp:update-filter)
    |     |     +--rw (yp:update-filter)?
    |     |        +--:(yp:subtree)
    |     |        |  +--rw yp:subtree-filter?
    |     |        +--:(yp:xpath)
    |     |           +--rw yp:xpath-filter?        yang:xpath1.0
    |     +--rw receivers
    |     |  +--rw receiver* [address]
    |     |     +--rw address     inet:host
    |     |     +--rw port        inet:port-number
    |     |     +--rw protocol?   transport-protocol
    |     +--rw (notification-origin)?
    |     |  +--:(interface-originated)
    |     |  |  +--rw source-interface?       if:interface-ref
    |     |  +--:(address-originated)
```

```
      |     |     +--rw source-vrf?             string
      |     |     +--rw source-address?         inet:ip-address-no-zone
      |     +--rw (yp:update-trigger)?
      |     |  +--:(yp:periodic)
      |     |  |  +--rw yp:period               yang:timeticks
      |     |  |  +--rw yp:anchor-time?         yang:date-and-time
      |     |  +--:(yp:on-change) {on-change}?
      |     |     +--rw yp:dampening-period     yang:timeticks
      |     |     +--rw yp:no-synch-on-start?   empty
      |     |     +--rw yp:excluded-change*     change-type
      |     +--rw yp:dscp?                inet:dscp
      |     +--rw yp:weighting?           uint8
      |     +--rw yp:dependency?          sn:subscription-id
      +--ro subscriptions
         +--ro subscription* [identifier]
            +--ro identifier                  subscription-id
            +--ro configured-subscription?
            |                        empty {configured-subscriptions}?
            +--ro stream?                     stream
            +--ro encoding?                   encoding
            +--ro replay-start-time?          yang:date-and-time
            +--ro stop-time?                  yang:date-and-time
            +--ro (filter-type)?
            |  +--:(by-reference)
            |  |  +--ro filter-ref?                filter-ref
            |  +--:(event-filter)
            |  |  +--ro filter?
            |  +--:(yp:update-filter)
            |     +--ro (yp:update-filter)?
            |        +--:(yp:subtree)
            |        |  +--ro yp:subtree-filter?
            |        +--:(yp:xpath)
            |           +--ro yp:xpath-filter?           yang:xpath1.0
            +--ro (notification-origin)?
            |  +--:(interface-originated)
            |  |  +--ro source-interface?          if:interface-ref
            |  +--:(address-originated)
            |     +--ro source-vrf?              string
            |     +--ro source-address?          inet:ip-address-no-zone
            +--ro receivers
            |  +--ro receiver* [address]
            |     +--ro address                   inet:host
            |     +--ro port                      inet:port-number
            |     +--ro protocol?                 transport-protocol
            |     +--ro pushed-notifications?     yang:counter64
            |     +--ro excluded-notifications?   yang:counter64
            +--ro subscription-status?        subscription-status
            +--ro (yp:update-trigger)?
```

```
            |  +--:(yp:periodic)
            |  |  +--ro yp:period                 yang:timeticks
            |  |  +--ro yp:anchor-time?           yang:date-and-time
            |  +--:(yp:on-change) {on-change}?
            |     +--ro yp:dampening-period       yang:timeticks
            |     +--ro yp:no-synch-on-start?     empty
            |     +--ro yp:excluded-change*       change-type
            +--ro yp:dscp?                   inet:dscp
            +--ro yp:weighting?              uint8
            +--ro yp:dependency?             sn:subscription-id

    rpcs:
      +---x establish-subscription
      |  +---w input
      |  |  +---w stream?                stream
      |  |  +---w encoding?              encoding
      |  |  +---w replay-start-time?     yang:date-and-time
      |  |  +---w stop-time?             yang:date-and-time
      |  |  +---w (filter-type)?
      |  |  |  +--:(by-reference)
      |  |  |  |  +---w filter-ref?            filter-ref
      |  |  |  +--:(event-filter)
      |  |  |  |  +---w filter?
      |  |  |  +--:(yp:update-filter)
      |  |  |     +---w (yp:update-filter)?
      |  |  |        +--:(yp:subtree)
      |  |  |        |  +---w yp:subtree-filter?
      |  |  |        +--:(yp:xpath)
      |  |  |           +---w yp:xpath-filter?       yang:xpath1.0
      |  |  +---w (yp:update-trigger)?
      |  |  |  +--:(yp:periodic)
      |  |  |  |  +---w yp:period              yang:timeticks
      |  |  |  |  +---w yp:anchor-time?        yang:date-and-time
      |  |  |  +--:(yp:on-change) {on-change}?
      |  |  |     +---w yp:dampening-period    yang:timeticks
      |  |  |     +---w yp:no-synch-on-start?  empty
      |  |  |     +---w yp:excluded-change*    change-type
      |  |  +---w yp:dscp?               inet:dscp
      |  |  +---w yp:weighting?          uint8
      |  |  +---w yp:dependency?         sn:subscription-id
      |  +--ro output
      |     +--ro subscription-result         subscription-result
      |     +--ro (result)?
      |        +--:(no-success)
      |        |  +--ro filter-failure?             string
      |        |  +--ro replay-start-time-hint?     yang:date-and-time
      |        |  +--ro yp:period-hint?             yang:timeticks
      |        |  +--ro yp:error-path?              string
```

```
|        |  +--ro yp:object-count-estimate?   uint32
|        |  +--ro yp:object-count-limit?      uint32
|        |  +--ro yp:kilobytes-estimate?      uint32
|        |  +--ro yp:kilobytes-limit?         uint32
|        +--:(success)
|           +--ro identifier                  subscription-id
+---x modify-subscription
|  +---w input
|  |  +---w identifier?            subscription-id
|  |  +---w stop-time?             yang:date-and-time
|  |  +---w (filter-type)?
|  |  |  +--:(by-reference)
|  |  |  |  +---w filter-ref?            filter-ref
|  |  |  +--:(event-filter)
|  |  |  |  +---w filter?
|  |  |  +--:(yp:update-filter)
|  |  |     +---w (yp:update-filter)?
|  |  |        +--:(yp:subtree)
|  |  |        |  +---w yp:subtree-filter?
|  |  |        +--:(yp:xpath)
|  |  |           +---w yp:xpath-filter?       yang:xpath1.0
|  |  +---w (yp:update-trigger)?
|  |     +--:(yp:periodic)
|  |     |  +---w yp:period              yang:timeticks
|  |     |  +---w yp:anchor-time?        yang:date-and-time
|  |     +--:(yp:on-change) {on-change}?
|  |        +---w yp:dampening-period    yang:timeticks
|  +--ro output
|     +--ro subscription-result         subscription-result
|     +--ro (result)?
|        +--:(no-success)
|           +--ro filter-failure?             string
|           +--ro yp:period-hint?             yang:timeticks
|           +--ro yp:error-path?              string
|           +--ro yp:object-count-estimate?   uint32
|           +--ro yp:object-count-limit?      uint32
|           +--ro yp:kilobytes-estimate?      uint32
|           +--ro yp:kilobytes-limit?         uint32
+---x delete-subscription
|  +---w input
|  |  +---w identifier    subscription-id
|  +--ro output
|     +--ro subscription-result    subscription-result
+---x kill-subscription
   +---w input
   |  +---w identifier    subscription-id
   +--ro output
      +--ro subscription-result    subscription-result
```

```
    notifications:
      +---n replay-complete
      |  +--ro identifier    subscription-id
      +---n notification-complete
      |  +--ro identifier    subscription-id
      +---n subscription-started
      |  +--ro identifier              subscription-id
      |  +--ro stream?                 stream
      |  +--ro encoding?               encoding
      |  +--ro replay-start-time?      yang:date-and-time
      |  +--ro stop-time?              yang:date-and-time
      |  +--ro (filter-type)?
      |  |  +--:(by-reference)
      |  |  |  +--ro filter-ref?             filter-ref
      |  |  +--:(event-filter)
      |  |  |  +--ro filter?
      |  |  +--:(yp:update-filter)
      |  |     +--ro (yp:update-filter)?
      |  |        +--:(yp:subtree)
      |  |        |  +--ro yp:subtree-filter?
      |  |        +--:(yp:xpath)
      |  |           +--ro yp:xpath-filter?        yang:xpath1.0
      |  +--ro (yp:update-trigger)?
      |  |  +--:(yp:periodic)
      |  |  |  +--ro yp:period               yang:timeticks
      |  |  |  +--ro yp:anchor-time?         yang:date-and-time
      |  |  +--:(yp:on-change) {on-change}?
      |  |     +--ro yp:dampening-period     yang:timeticks
      |  |     +--ro yp:no-synch-on-start?   empty
      |  |     +--ro yp:excluded-change*     change-type
      |  +--ro yp:dscp?                inet:dscp
      |  +--ro yp:weighting?           uint8
      |  +--ro yp:dependency?          sn:subscription-id
      +---n subscription-resumed
      |  +--ro identifier    subscription-id
      +---n subscription-modified
      |  +--ro identifier              subscription-id
      |  +--ro stream?                 stream
      |  +--ro encoding?               encoding
      |  +--ro replay-start-time?      yang:date-and-time
      |  +--ro stop-time?              yang:date-and-time
      |  +--ro (filter-type)?
      |  |  +--:(by-reference)
      |  |  |  +--ro filter-ref?             filter-ref
      |  |  +--:(event-filter)
      |  |  |  +--ro filter?
      |  |  +--:(yp:update-filter)
      |  |     +--ro (yp:update-filter)?
```

```
      |  |        +--:(yp:subtree)
      |  |        |  +--ro yp:subtree-filter?
      |  |        +--:(yp:xpath)
      |  |           +--ro yp:xpath-filter?       yang:xpath1.0
      |  +--ro (yp:update-trigger)?
      |  |  +--:(yp:periodic)
      |  |  |  +--ro yp:period              yang:timeticks
      |  |  |  +--ro yp:anchor-time?        yang:date-and-time
      |  |  +--:(yp:on-change) {on-change}?
      |  |     +--ro yp:dampening-period    yang:timeticks
      |  |     +--ro yp:no-synch-on-start?  empty
      |  |     +--ro yp:excluded-change*    change-type
      |  +--ro yp:dscp?                inet:dscp
      |  +--ro yp:weighting?           uint8
      |  +--ro yp:dependency?          sn:subscription-id
      +---n subscription-terminated
      |  +--ro identifier        subscription-id
      |  +--ro error-id          subscription-errors
      |  +--ro filter-failure?   string
      +---n subscription-suspended
         +--ro identifier        subscription-id
         +--ro error-id          subscription-errors
         +--ro filter-failure?   string
  module: ietf-yang-push

    notifications:
      +---n push-update
      |  +--ro subscription-id       sn:subscription-id
      |  +--ro time-of-update?       yang:date-and-time
      |  +--ro updates-not-sent?     empty
      |  +--ro datastore-contents?
      +---n push-change-update {on-change}?
         +--ro subscription-id      sn:subscription-id
         +--ro time-of-update?      yang:date-and-time
         +--ro updates-not-sent?    empty
         +--ro datastore-changes?
```

Figure 11: Model structure

Selected components of the model are summarized in the following subsections.

4.2. Filters

Filters can be supported via a reference to an entry in the filter container, or via direct embedding within a subscription itself. When a reference is used, it becomes possible to configure filters independently of the lifecycle of a subscription. This facilitates

the reuse of filter definitions, which can be important in case of complex filter conditions. Referenced filters can also allow an implementation to avoid evaluating filter acceptability during a dynamic subscription request.

Whether referenced or in-line, filters used for yang-push must be of case update-filters, and must follow the syntax and semantics of RFC 6241. It is not expected that implementations will support comprehensive XPATH syntax and boundless complexity. It will be up to implementations to describe what is viable, but the goal is to provide equivalent capabilities to what is available with a GET. Yang-push implementations must reject dynamic subscriptions or suspend configured subscriptions if they include filters which are unsupportable on a platform.

It is conceivable that other types of filters will be introduced for yang-push in the future. To support such filter types, additional filter cases can augment the data model.

4.3. Subscription configuration

Both configured and dynamic subscriptions are represented within the list subscription-config. Each subscription has own list elements. New and enhanced parameters extending the basic subscription data model in [I-D:netconf-sub-notif] include:

- An update filter identifying yang nodes of interest. Filter contents are specified via a reference to an existing filter, or via an in-line definition for only that subscription. The case statement differentates the options.

- For periodic subscriptions, triggered updates will occur at the boundaries of a specified time interval. The periodic parameters which define this interval include:

  * a "period" which defines duration between period push updates.

  * an "anchor-time". Update intervals always fall on the points in time that are a multiple of a period after the anchor time. If anchor time is not provided, then the anchor time must be set for when the initial push update can be sent.

- When used in conjunction with period, the boundaries of periodic update periods may be calculated.

- For on-change subscriptions, assuming the dampening period has completed, triggered occurs whenever a change in the subscribed

information is detected. On-change subscriptions have more complex semantics that is guided by its own set of parameters:

* a "dampening-period" specifies the interval that must pass before a successive update for the subscription 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 for this subscription has passed.

* an "excluded-change" flag which allows restriction of the types of changes for which updates should be sent (changes to object values, object creation or deletion events).

* a "no-synch-on-start" flag which specifies whether a complete update with all the subscribed data should be sent at the beginning of a subscription.

- Optional qos parameters to indicate the treatment of a susbcription relative to other traffic between publisher and receiver. These include:

  * A "dscp" QoS marking which should be stamped on packets to show network QoS treatment.

  * A "weighting" so that bandwidth proportional to this weighting can be allocated to this subscription relative to others for that receiver.

  * a "dependency" upon another subscription. No push should be sent until all updates for the referenced subscription have been queued and sent.

- A subscription’s weighting should work identically to stream dependency weighting as described within RFC 7540, section 5.3.2.

- A subscription’s dependency should work identically to stream dependency as described within RFC 7540, sections 5.3.1, 5.3.3, and 5.3.4. If a dependency is attempted via an RPC, but the referenced subscription does not exist, the dependency will be removed.

4.4. Notifications

4.4.1. Monitoring and OAM Notifications

OAM notifications and mechanism are with one exception reused from [I-D:netconf-sub-notif].

The one excpetion is the excluded-notifications object is not applicable for yang-push. This is because discarded notifications for datastore does not have meaning in this context, and should always be zero.

4.4.2. 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" on a complete set of susbcribed datastore contents. This will be done the start of an on-change subscription, unless no-synch-on-start is specified for that subscription. In addition, this notification MAY be used during the subscription. This might be a useful thing to do if change updates were not sent as expected (as indicated by the "updates-not-sent" flag, or an identification of loss in pushed updates), or for general resynchronization of a datastore extract at longer period intervals (such as once per day) to mitigate the possibility of any application-dependent synchronization drift. A mandatory requirement defining when to sending a push-update notification in conjunction with on-change subscription is not asserted in this specification beyond synch-on-start. However an on-change receiver must be able to handle an unsolicited push-update as a state synchonization reset.

The format and syntax of the contained update notification 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).

If the application detects a discontinuity in the updates it is pushing, 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.

4.5. RPCs

YANG-Push subscriptions are established, modified, and deleted using RPCs augmented from [I-D:netconf-sub-notif].

4.5.1. Establish-subscription RPC

The subscriber sends an establish-subscription RPC with the parameters in section 3.1. An example might look like:

```
<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">
      <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 12: 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 13: 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 information indicates the node is unavailable. 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">
       subtree-unavailable
    </subscription-result>
    <filter-failure
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       /ex:foo
    </filter-failure>
</rpc-reply>
```

Figure 14: 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 15: 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">
       period-unsupported
    </subscription-result>
    <period-hint>100</period-hint>
</rpc-reply>
```

Figure 16: Establish-subscription error response example 2

4.5.2. 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>
      <period>250</period>
   </modify-subscription>
</netconf:rpc>
```

Figure 17: 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>
</rpc-reply>
```

Figure 18: 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.

4.5.3. 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">
   <subscription-result
      xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
         ok
   </subscription-result>
</rpc-reply>
```

Figure 19: Delete subscription

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

4.5.4. 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 [RFC8040] 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 module implemented by the publisher. 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.

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@2017-02-08.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-subscribed-notifications {
    prefix sn;
  }

  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:ludwig@clemm.org>

     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 2017-02-08 {
    description
      "Updates to simplify modify-subscription, add anchor-time";
    reference
      "YANG Datastore Push, draft-ietf-netconf-yang-push-05";
  }

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

 /*
  * IDENTITIES
  */

  /* Additional errors for subscription operations */
  identity period-unsupported {
    base sn:error;
    description
      "Requested time period is too short. This can be for both
       periodic and on-change dampening.";
  }

  identity qos-unsupported {
    base sn:error;
    description
      "Subscription QoS parameters not supported on this platform.";
  }

  identity dscp-unavailable {
    base sn:error;
```

```
    description
      "Requested DSCP marking not allocatable.";
  }

  identity on-change-unsupported {
    base sn:error;
    description
      "On-change not supported.";
  }

  identity synch-on-start-unsupported {
    base sn:error;
    description
      "On-change synch-on-start not supported.";
  }

  identity synch-on-start-datatree-size {
    base sn:error;
    description
      "Synch-on-start would push a datatree which exceeds size limit.";
  }

  identity reference-mismatch {
    base sn:error;
    description
      "Mismatch in filter key and referenced yang subtree.";
  }

  identity subtree-unavailable {
    base sn:error;
    description
      "Referenced yang subtree doesn't exist, or is a node where read
       access is not permitted.";
  }

  identity datatree-size {
    base sn:error;
    description
      "Resulting push updates would exceed size limit.";
  }

  /* Additional types of streams */
  identity yang-push {
    base sn:stream;
    description
      "A conceptual datastream consisting of all datastore updates,
       including operational and configuration data.";
  }
```

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

  /* Additional transport option */
  identity http2 {
    base sn: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.";
  }

  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.";
          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-modifiable {
    description
      "This grouping describes the datastore specific subscription
       conditions that can be changed during the lifetime of the
       subscription.";
    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 as 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 subset as defined by a filter.";
        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 for the subscription.";
        }
      }
    }
  }

  grouping update-policy {
    description
      "This grouping describes the datastore specific subscription
       conditions of a subscription.";
    uses update-policy-modifiable {
      augment "update-trigger/on-change" {
        description
          "Includes objects not modifiable once subscription is
           established.";
        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-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 update-qos {
    description
      "This grouping describes Quality of Service information
       concerning a subscription.  This information is passed to lower
       layers for transport prioritization and treatment";
    leaf dscp {
      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 weighting {
      type uint8 {
         range "0 .. 255";
      }
      description
        "Relative weighting for a subscription. Allows an underlying
         transport layer perform informed load balance allocations
         between various subscriptions";
      reference
```

```
        "RFC-7540, section 5.3.2";
    }
    leaf dependency {
      type sn:subscription-id;
      description
        "Provides the Subscription ID of a parent subscription which has
         absolute priority should that parent have push updates ready to
         egress the publisher. In other words, there should be no
         streaming of objects from the current subscription if of the
         parent has something ready to push.";
      reference
        "RFC-7540, section 5.3.1";
    }
  }

  grouping update-error-hints {
    description
      "Allow return additional negotiation hints that apply
       specifically to push updates.";
    leaf period-hint {
      type yang:timeticks;
      description
        "Returned when the requested time period is too short. This hint
         can assert an viable period for both periodic push cadence and
         on-change dampening.";
    }
    leaf error-path {
      type string;
      description
        "Reference to a YANG path which is associated with the error
         being returned.";
    }
    leaf object-count-estimate {
      type uint32;
      description
        "If there are too many objects which could potentially be
         returned by the filter, this identifies the estimate of the
         number of objects which the filter would potentially pass.";
    }
    leaf object-count-limit {
      type uint32;
      description
        "If there are too many objects which could be returned by the
         filter, this identifies the upper limit of the publisher's
         ability to service for this subscription.";
    }
    leaf kilobytes-estimate {
      type uint32;
```

```
      description
        "If the returned information could be beyond the capacity of the
         publisher, this would identify the data size which could result
         from this filter.";
    }
    leaf kilobytes-limit {
      type uint32;
      description
        "If the returned information would be beyond the capacity of the
         publisher, this identifies the upper limit of the publisher's
         ability to service for this subscription.";
    }
  }

  augment "/sn:establish-subscription/sn:input" {
    description
      "Define additional subscription parameters that apply
       specifically to push updates";
    uses update-policy;
    uses update-qos;
  }
  augment "/sn:establish-subscription/sn:input/"+
          "sn: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 "/sn:establish-subscription/sn:output/"+
    "sn:result/sn:no-success" {
    description
      "Add push datastore error info and hints to RPC output.";
    uses update-error-hints;
  }
  augment "/sn:modify-subscription/sn:input" {
    description
      "Define additional subscription parameters that apply
       specifically to push updates.";
    uses update-policy-modifiable;
  }
  augment "/sn:modify-subscription/sn:input/"+
          "sn: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 "/sn:modify-subscription/sn:output/"+
    "sn:result/sn:no-success" {
    description
      "Add push datastore error info and hints to RPC output.";
    uses update-error-hints;
  }

  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 sn: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 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 sn: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 occurred 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 "/sn: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 update-qos;
  }
  augment "/sn:subscription-started/sn: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 "/sn: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 update-qos;
  }
  augment "/sn:subscription-modified/sn: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 "/sn:filters/sn:filter/"+
          "sn: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 "/sn:subscription-config/sn:subscription" {
    description
      "Contains the list of subscriptions that are configured,
       as opposed to established via RPC or other means.";
    uses update-policy;
    uses update-qos;
  }
  augment "/sn:subscription-config/sn:subscription/"+
          "sn:filter-type" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      uses update-filter;
    }
  }
  augment "/sn:subscriptions/sn: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 update-qos;
  }
  augment "/sn:subscriptions/sn:subscription/"+
          "sn: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.

For both configured and dynamic subscriptions it is essential to authenticate and authorize that receiver via some transport level mechanism before sending any push updates.

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:netconf-sub-notif]
Clemm, A., Gonzalez Prieto, A., Voit, E., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG-Defined Event Notifications", draft-ietf-netconf-subscribed-notifications-00 (work in progress), February 2017.

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <http://www.rfc-editor.org/info/rfc6470>.

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

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <http://www.rfc-editor.org/info/rfc7895>.

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

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <http://www.rfc-editor.org/info/rfc7951>.

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <http://www.rfc-editor.org/info/rfc8072>.

8.2. Informative References

[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.

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

Appendix A. Technologies to be considered for future iterations

A.1. Proxy YANG Subscription when the Subscriber and Receiver are different

The properties of Dynamic and Configured Subscriptions can be combined to enable deployment models where the Subscriber and Receiver are different. Such separation can be useful with some combination of:

- An operator does not want the subscription to be dependent on the maintenance of transport level keep-alives. (Transport independence provides different scalability characteristics.)
- There is not a transport session binding, and a transient Subscription needs to survive in an environment where there is unreliable connectivity with the Receiver and/or Subscriber.
- An operator wants the Publisher to include highly restrictive capacity management and Subscription security mechanisms outside of domain of existing operational or programmatic interfaces.

To build a Proxy Subscription, first the necessary information must be signaled as part of the <establish-subscription>. Using this set of Subscriber provided information; the same process described within section 3 will be followed.

After a successful establishment, if the Subscriber wishes to track the state of Receiver subscriptions, it may choose to place a separate on-change Subscription into the "Subscriptions" subtree of the YANG Datastore on the Publisher.

A.2. OpState and Filters

Currently there are ongoing discusssions to revise the concept of datastores, allowing for proper handling and distinction of intended versus applied configurations and extending the notion of a datastore to operational data. When finalized, the new concept may open up the possibility for new types of subscription filters, for example, targeting specific datastores and targeting (potentially) differences in datatrees across different datastores.

Likewise, it is conceivable that filters are defined that apply to metadata, such as data nodes for which metadata has been defined that meets a certain criteria.

Defining any such subscription filters at this point would be highly speculative in nature. However, it should be noted that

corresponding extensions may be defined in future specifications. Any such extensions will be straightforward to accommodate by introducing a model that defines new filter types, and augmenting the new filter type into the subscription model.

A.3. Splitting push updates

Push updates may become fairly large and extend across multiple subsystems in a YANG-Push Server. As a result, it conceivable to not combine all updates into a single update message, but to split updates into multiple separate update messages. Such splitting could occur along multiple criteria: limiting the number of data nodes contained in a single update, grouping updates by subtree, grouping updates by internal subsystems (e.g., by line card), or grouping them by other criteria.

Splitting updates bears some resemblance to fragmenting packets. In effect, it can be seen as fragmenting update messages at an application level. However, from a transport perspective, splitting of update messages is not required as long as the transport does not impose a size limitation or provides its own fragmentation mechanism if needed. We assume this to be the case for YANG-Push. In the case of NETCONF, RESTCONF, HTTP/2, no limit on message size is imposed. In case of other transports, any message size limitations need to be handled by the corresponding transport mapping.

There may be some scenarios in which splitting updates might still make sense. For example, if updates are collected from multiple independent subsystems, those updates could be sent separately without need for combining. However, if updates were to be split, other issues arise. Examples include indicating the number of updates to the receiver, dinstinguishing a missed fragment from a missed update, and the ordering with which updates are received. Proper addressing those issues would result in considerable complexity, while resulting in only very limited gains. In addition, if a subscription is found to result in updates that are too large, a publisher can always reject the request for a subscription while the subscriber is always free to break a subscription up into multiple subscriptions.

A.4. Potential Subscription Parameters

A possible is 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.

Appendix B. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

Issue #6: Data plane notifications and layered headers. Specifically how do we want to enable standard header unificiation and bundle support vs. the data plane notifications currently defined.

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v04 to v05

- Referenced based subscription document changed to Subscribed Notifications from 5277bis.
- Getting operational data from filters
- Extension notifiable-on-change added
- New appendix on potential futures. Moved text into there from several drafts.
- Subscription configuration section now just includes changed parameters from Subscribed Notifications
- Subscription monitoring moved into Subscribed Notifications
- New error and hint mechanisms included in text and in the yang model.
- Updated examples based on the error definitions
- Groupings updated for consistency
- Text updates throughout

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
Huawei

Email: ludwig@clemm.org

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