NETCONF
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2018

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

February 23, 2018

YANG Datastore Subscription
draft-ietf-netconf-yang-push-15

Abstract

Via the mechanism described in this document, subscriber applications may request a continuous, customized stream of updates from a YANG datastore. Providing such visibility into changes made upon YANG configuration and operational objects enables new capabilities based on the remote mirroring of configuration and operational state.

Table of Contents

1. Introduction

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

- Polling incurs significant latency. This latency prohibits many application types.

- Polling cycles may be missed, requests may be delayed or get lost, often when the network is under stress and the need for the data is the greatest.

- Polling requests may undergo slight fluctuations, resulting in intervals of different lengths. The resulting data is difficult to calibrate and compare.

- For applications that monitor for changes, many remote polling cycles place ultimately fruitless load on the network, devices, and applications.

A more effective alternative to polling is for an application to receive automatic and continuous updates from a targeted subset of a datastore. Accordingly, there is a need for a service that allows applications to subscribe to updates from a datastore and that enables the publisher to push and in effect stream those updates. The requirements for such a service have been documented in [RFC7923].

This document defines a corresponding solution that is built on top of "Custom Subscription to Event Streams" [I-D.draft-ietf-netconf-subscribed-notifications]. Supplementing that work are YANG data model augmentations, extended RPCs, and new

datastore specific update notifications. Transport options for [I-D.draft-ietf-netconf-subscribed-notifications] will work seamlessly with this solution.

2. Definitions and Acronyms

The terms below supplement those defined in [I-D.draft-ietf-netconf-subscribed-notifications]. In addition, the term "datastore" is defined in [I-D.draft-ietf-netmod-revised-datastores].

Datastore node: An instance of management information in a datastore. Also known as "object".

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

Datastore subtree: An instantiated datastore node and the datastore nodes that are hierarchically contained within it.

Update record: A representation datastore node update(s) resulting from the application of a selection filter for a subscription. An update record will include the value/property of one or more datastore nodes at a point in time. It may contain the update type for each datastore node (e.g., add, change, delete). Also included may be metadata/headers such as a subscription identifier.

Selection filter: Evaluation and/or selection criteria, which may be applied against a targeted set of objects.

Update trigger: A mechanism that determines when an update record needs to be generated.

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

3. Solution Overview

This document specifies a solution for a push update subscription service. This solution supports dynamic as well as configured subscriptions to information updates from datastores. Subscriptions specify when notification messages should be sent and what data to include in update records. 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 enhances the subscription model defined in [I-D.draft-ietf-netconf-subscribed-notifications] with capabilities that allow subscribers to subscribe to datastore node updates, specifically to specify the triggers defining when to generate update records as well as what to include in an update record. Key enhancements include:

- Specification of selection filters which identify targeted YANG datastore nodes and/or subtrees within a datastore for which updates are to be pushed.

- Specification of update policies contain conditions which trigger the generation and pushing of new update records. There are two types of triggers for subscriptions: periodic and on-change.

  * For periodic subscriptions, the trigger is specified by two parameters that define when updates are to be pushed. These parameters are the period interval with which to report updates, and an anchor time which can be used to calculate at which point in time updates need to be assembled and sent.

  * For on-change subscriptions, a trigger occurs whenever a change in the subscribed information is detected. Included are additional parameters such as:

    + Dampening period: In an on-change subscription, detected object changes should be sent as quickly as possible. However it may be undesirable to send a rapid series of object changes. Such behavior has the potential to exhaust of resources in the publisher or receiver. In order to protect against that, a dampening period MAY be used to specify the interval which must pass before successive update records for the same subscription are generated for a receiver. The dampening period collectively applies to the set of all datastore nodes selected by a single subscription and sent to a single receiver. This means that when there is a change to one or more subscribed objects, an update record containing those objects is created either immediately when no dampening period is in effect, or at the end of a dampening period. If multiple changes to a single object occur during a dampening period, only the value that is in effect at the time the update record is created is included. The dampening period goes into effect every time an update record completes assembly.

+ Change type: This parameter can be used to reduce the types of datastore changes for which updates are sent (e.g., you might only send when an object is created or deleted, but not when an object value changes).

+ No Synch on start: defines whether or not a complete push-update of all subscribed data will be sent at the beginning of a subscription. Such early synchronization establishes the frame of reference for subsequent updates.

o An encoding (using anydata) for the contents of periodic and on-change push updates.

3.2. Negotiation of Subscription Policies

A dynamic subscription request SHOULD be declined if a publisher's assessment is that it may be unable to provide update records meeting the terms of an "establish-subscription" or "modify-subscription" rpc request. In this case, a subscriber may quickly follow up with a new rpc request using different parameters.

Random guessing at different parameters by a subscriber is to be discouraged. Therefore, in order to minimize the number of subscription iterations between subscriber and publisher, dynamic subscription supports a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is in the form of supplemental information which may be inserted within error responses to a failed rpc request. This returned error response information, when considered, should increase the likelihood of success for subsequent rpc requests. Such hints include suggested periodic time intervals, acceptable dampening periods, and size estimates for the number or objects which would be returned from a proposed selection filter. However, there are no guarantees that subsequent requests which consider these hints will be accepted.

3.3. On-Change Considerations

On-change subscriptions allow subscribers to receive updates whenever changes to targeted objects occur. As such, on-change subscriptions are particularly effective for data that changes infrequently, yet for which applications need to be quickly notified whenever a change does occur with minimal delay.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

Whether or not to accept or reject on-change subscription requests when the scope of the subscription contains objects for which on-change is not supported is up to the publisher implementation. A publisher MAY accept an on-change subscription even when the scope of the subscription contains objects for which on-change is not supported. In that case, updates are sent only for those objects within the scope that do support on-change updates whereas other objects are excluded from update records, whether or not their values actually change. In order for a subscriber to determine whether objects support on-change subscriptions, objects are marked accordingly on a publisher. Accordingly, when subscribing, it is the responsibility of the subscriber to ensure it is aware of which objects support on-change and which do not. For more on how objects are so marked, see Section 3.10.

Alternatively, a publisher MAY decide to simply reject an on-change subscription in case the scope of the subscription contains objects for which on-change is not supported. In case of a configured subscription, the subscription MAY be suspended.

To avoid flooding receivers with repeated updates for subscriptions containing fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update record for a given object is generated, no other updates for this particular subscription will be created 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 include those which were deleted or newly created during that dampening period. If an object has returned to its original value (or even has been created and then deleted) during the dampening-period, the last change will still be sent. This will indicate churn is occurring on that object.

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

Putting it all together, following is the conceptual process for creating an push-change-update notification:

1. Just before a change, or at the start of a dampening period, evaluate any filtering and any access control rules. The result is a set "A" of datastore nodes and subtrees.

2. Just after a change, or at the end of a dampening period, evaluate any filtering and any (possibly new) access control rules. The result is a set "B" of datastore nodes and subtrees.

3. Construct a YANG patch record for going from A to B.

4. If there were any changes made between A and B which canceled each other out, insert into the YANG patch record the last change made for any object which otherwise wouldn't have appeared.

5. If the resulting patch record is non-empty, send it to the receiver.

Note: In cases where a subscriber wants to have separate dampening periods for different objects, multiple subscriptions with different objects in a selection filter can be created.

3.4. Promise-Theory Considerations

A subscription to updates from a datastore is intended to obviate the need for polling. However, in order to do so, it is critical that subscribers can rely on the subscription and have confidence that they will indeed receive the subscribed updates without having to worry about updates being silently dropped. In other words, a subscription constitutes a promise on the side of the publisher to provide the receivers with updates per the terms of the subscription.

Now, there are many reasons why a publisher may at some point no longer be able to fulfill the terms of the subscription, even if the subscription had been entered into with good faith. For example, the volume of data objects may be larger than anticipated, the interval may prove too short to send full updates in rapid succession, or an internal problem may prevent objects from being collected. If for some reason the publisher of a subscription is not able to keep its promise, receivers MUST be notified immediately and reliably. The publisher MAY also suspend the subscription.

A publisher SHOULD reject a request for a subscription if it is unlikely that the publisher will be able fulfill the terms of that subscription request. In such cases, it is preferable to have a subscriber request a less resource intensive subscription than to deal with frequently degraded behavior.

3.5. Data Encodings

3.5.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been read using a retrieval operation.

3.5.2. On-Change Subscriptions

In an on-change subscription, updates need to indicate not only values of changed datastore nodes but also the types of changes that occurred since the last update. Therefore encoding rules for data in on-change updates will generally follow YANG-patch operation as specified in [RFC8072]. The YANG-patch will describe what needs to be applied to the earlier state reported by the preceding update, to result in the now-current state. Note that contrary to [RFC8072], objects encapsulated are not restricted to configuration objects only.

However a patch must be able to do more than just describe the delta from the previous state to the current state. As per Section 3.3, it must also be able to identify if transient changes have occurred on an object during a dampening period. To support this, it is valid to encode a YANG patch operation so that its application would result in no change between the previous and current state. This indicates that some churn has occurred on the object. An example of this would be a patch that does a "create" operation for a datastore node where the receiver believes one already exists, or a "merge" operation which replaces a previous value with the same value. Note that this means that the "create" and "delete" errors described in [RFC8072] section 2.5 are not errors, and are valid operations with YANG push.

3.6. Defining the Selection with a Datastore

A subscription must specify both the selection filters and the datastore against which these selection filters will be applied. This information is used to choose and subsequently push data from the publisher's datastore to the receivers.

Only a single selection filter can be applied to a subscription at a time. An rpc request proposing a new selection filter MUST remove any existing filter. The following selection filter types are included in the yang-push data model, and may be applied against a datastore:

- subtree: A subtree selection filter identifies one or more datastore subtrees. When specified, update records will only come from the datastore nodes of selected datastore subtree(s). The

syntax and semantics correspond to that specified for [RFC6241] section 6.

- o xpath: An xpath selection filter is an XPath expression that returns a node set. When specified, updates will only come from the selected data nodes.

These filters are intended to be used as selectors that define which objects are within the scope of a subscription. A publisher MUST support at least one type of selection filter.

Xpath itself provides powerful filtering constructs and care must be used in filter definition. As an example, consider an xpath filter with a boolean result; such a result will not provide an easily interpretable subset of a datastore. Beyond the boolean example, it is quite possible to define an xpath filter where results are easy for an application to misinterpret. Consider an xpath filter which only passes a datastore object when an interface is up. It is up to the receiver to understand implications of the presence or absence of objects in each update.

When the set of selection filtering criteria is applied for a periodic subscription, all selected datastore nodes to which a receiver has access are provided to that receiver. If the same filtering criteria is applied to an on-change subscription, only the subset of those datastore nodes supporting on-change is provided. A datastore node which doesn't support on-change is never sent as part of an on-change subscription's "push-update" or "push-change-update".

3.7. Streaming Updates

Contrary to traditional data retrieval requests, datastore subscription enables an unbounded series of update records to be streamed over time. Two generic YANG notifications for update records have been defined for this: "push-update" and "push-change-update".

A "push-update" notification defines a complete, filtered update of the datastore per the terms of a subscription. This type of YANG notification is used for continuous updates of periodic subscriptions. A "push-update" notification can also be used 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 an on-change subscription. The "push-update" update record contains an instantiated datastore subtree with all of the subscribed contents. The content of the update record is equivalent to the contents that would be obtained

had the same data been explicitly retrieved using a datastore retrieval operation using the same transport with the same filters applied.

A "push-change-update" notification is the most common type of update for on-change subscriptions. The update record in this case contains the set of changes that datastore nodes have undergone since the last notification message. In other words, this indicates which datastore nodes have been created, deleted, or have had changes to their values. In cases where multiple changes have occurred and the object has not been deleted, the object's most current value is reported. (In other words, for each object, only one change is reported, not its entire history. Doing so would defeat the purpose of the dampening period.)

These new "push-update" or "push-change-update" are encoded and placed within notification messages, and ultimately queued for egress over the specified transport.

The following is an example of a notification message for a subscription tracking the operational status of a single Ethernet port (per [RFC7223]). This notification message is encoded XML over NETCONF as per [I-D.draft-ietf-netconf-netconf-event-notifications].

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2017-10-25T08:00:11.22Z</eventTime>
 <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <subscription-id>1011</subscription-id>
   <datastore-contents>
     <interfaces-state xmlns="http://foo.com/ietf-interfaces">
       <interface>
         <name>eth0</name>
         <oper-status>up</oper-status>
       </interface>
     </interfaces-state>
   </datastore-contents>
 </push-update>
</notification>
```

Figure 1: Push example

The following is an example of an on-change notification message for the same subscription.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2017-10-25T08:22:33.44Z</eventTime>
 <push-change-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <subscription-id>89</subscription-id>
   <datastore-changes>
     <yang-patch xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-patch">
       <patch-id>1</patch-id>
       <edit>
         <edit-id>edit1</edit-id>
         <operation>merge</operation>
         <target>/ietf-interfaces:interfaces-state</target>
         <value>
           <interfaces-state xmlns="http://foo.com/ietf-interfaces">
             <interface>
               <name>eth0</name>
               <oper-status>down</oper-status>
             </interface>
           </interfaces-state>
         </value>
       </edit>
     </yang-patch>
   </datastore-changes>
 </push-change-update>
</notification>
```

Figure 2: Push example for on change

Of note in the above example is the 'patch-id' with a value of '1'. Per [RFC8072], the 'patch-id' is an arbitrary string. With YANG Push, the publisher SHOULD put into the 'patch-id' a counter starting at '1' which increments with every 'push-change-update' generated for a subscription. If used as a counter, this counter MUST be reset to '1' anytime a resynchronization occurs (i.e., with the sending of a 'push-update'). Also if used as a counter, the counter MUST be reset to '1' the after passing a maximum value of '99999'. Such a mechanism allows easy identification of lost or out-of-sequence update records.

3.8. Subscription Management

The RPCs defined within [I-D.draft-ietf-netconf-subscribed-notifications] have been enhanced to support datastore subscription negotiation. Included in these enhancements are error codes which can indicate why a datastore subscription attempt has failed.

A datastore subscription can be rejected for multiple reasons. This includes a too large subtree request, or the inability of the

publisher to push update records as frequently as requested. In such cases, no subscription is established. Instead, the subscription-result with the failure reason is returned as part of the RPC response. As part of this response, a set of alternative subscription parameters MAY be returned that would likely have resulted in acceptance of the subscription request. The subscriber may consider these as part of future subscription attempts.

The specific parameters to be returned in as part of the RPC error response depend on the specific transport that is used to manage the subscription. In the case of NETCONF [I-D.draft-ietf-netconf-netconf-event-notifications], the NETCONF RPC reply MUST include an "rpc-error" element with the following additional elements:

- "error-type" of "application".
- "error-tag" of "operation-failed".
- Optionally, an "error-severity" of "error" (this MAY but does not have to be included).
- "error-app-tag" with the value being a string that corresponds to an identity with a base of "establish-subscription-error" (for error responses to an establish-subscription request), "modify-subscription-error" (for error responses to a modify-subscription request), "delete-subscription-error" (for error responses to a delete-subscription request), "resynch-subscription-error" (for error responses to resynch-subscription request), or "kill-subscription-error" (for error responses to a kill-subscription request), respectively.
- In case of error responses to an establish-subscription or modify-subscription request: optionally, "error-info" containing XML-encoded data with hints regarding parameter settings that might lead to successful requests in the future, per yang-data definitions "establish-subscription-error-datastore" (for error responses to an establish-subscription request) or "modify-subscription-error-datastore (for error responses to a modify-subscription request), respectively. In case of an rpc error as a result of a delete-subscription, or a kill-subscription, or a resynch-subscription request, no error-info needs to be included, as the subscription-id is the only RPC input parameter and no hints regarding RPC input parameters need to be provided.

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-subscribed-notifications"
  xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <yp:datastore xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
   </yp:datastore>
    <yp:datastore-xpath-filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0">
      /ex:foo
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
  </establish-subscription>
</netconf:rpc>
```

Figure 3: Establish-Subscription example

the publisher might return:

```
 <rpc-reply message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-app-tag>period-unsupported</error-app-tag>
    <error-info>
      <establish-subscription-error-datastore
       xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
        <yp:period-hint>
          2000
        </yp:period-hint>
      </establish-subscription-error-datastore>
    </error-info>
   </rpc-error>
 </rpc-reply>
```

Figure 4: Error response example

3.9. Receiver Authorization

A receiver of subscription data MUST only be sent updates for which they have proper authorization. A publisher MUST ensure that no non-authorized data is included in push updates. To do so, it needs to

apply all corresponding checks applicable at the time of a specific pushed update and if necessary silently remove any non-authorized data from datastore subtrees. This enables YANG data pushed based on subscriptions to be authorized equivalently to a regular data retrieval (get) operation.

A publisher MUST allow for the possibility that a subscription's selection filter references non-existent or access-protected data. Such support permits a receiver the ability to monitor the entire lifecyle of some datastore tree. In this case, all "push-update" notifications must be sent empty, and no "push-change-update" notifications will be sent until some data becomes visible for a receiver.

A publisher MAY choose reject an establish-subscription request which selects non-existent or access-protected data. In addition, a publisher MAY choose to terminate a dynamic subscription or suspend a configured receiver when the authorization privileges of a receiver change, or the access controls for subscribed objects change. Such a capability enables the publisher to avoid having to support a continuous, and total filtering of an entire subscription's content.

In these cases above, the error identity "unchanging-selection" SHOULD be returned. This reduces the possibility of leakage of access controlled objects.

Each "push-update" and "push-change-update" MUST have access control applied. This includes validating that read access is permitted for any new objects selected since the last notification message was sent to a particular each receiver. To accomplish this, implementations SHOULD support the conceptual authorization model of [RFC6536bis], Section 3.2.4.

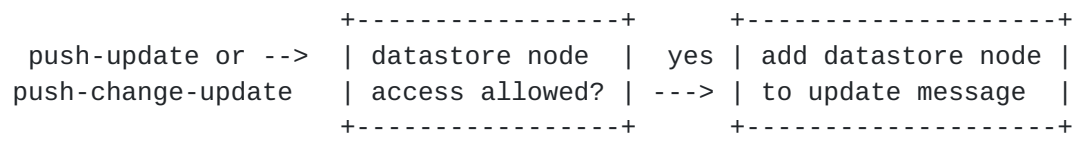

Figure 5: Updated [rfc6536bis] access control for push updates

If read access into previously accessible nodes has been lost due to a receiver permissions change, this SHOULD be reported as a patch "delete" operation for on-change subscriptions. If not capable of handling such receiver permission changes with such a "delete", publisher implementations MUST force dynamic subscription re-establishment or configured subscription re-initialization so that appropriate filtering is installed.

3.10. 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 datastore node changes frequently (e.g., [RFC7223]'s in-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 for a particular object.

In those cases, it will be important for client applications to have a way to identify for which objects on-change notifications are supported and for which ones they are not supported. Otherwise client applications will have no way of knowing whether they can indeed rely on their on-change subscription to provide them with the change updates that they are interested in. In other words, if implementations do not provide a solution and do not support comprehensive on-change notifiability, clients of those implementations will have no way of knowing what their on-change subscription actually covers.

Implementations are therefore strongly advised to provide a solution to this problem. It is expected that such a solution will be standardized at some point in the future. In the meantime and until this occurs, implementations will be expected to provide their own solution.

3.11. Other Considerations

3.11.1. Robustness and reliability

Particularly in the case of on-change updates, it is important that these updates do not get lost. Or in case the loss of an update is unavoidable, it is critical that the receiver is notified accordingly.

Update records for a single subscription MAY NOT be resequenced prior to transport.

It is conceivable that under certain circumstances, a publisher will recognize that it is unable to include within an update record 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 "incomplete-update" flag on any update record which is known to be missing information.

- It MAY choose to suspend a subscription as per [I-D.draft-ietf-netconf-subscribed-notifications].

- When resuming an on-change subscription, the publisher SHOULD generate a complete patch from the previous update record. If this is not possible and the "no-synch-on-start" option is not present for the subscription, then the full datastore contents MAY be sent via a "push-update" instead (effectively replacing the previous contents). If neither of these are possible, then an "incomplete-update" flag MUST be included on the next "push-change-update".

Note: It is perfectly acceptable to have a series of "push-change-update" notifications (and even "push update" notifications) serially queued at the transport layer awaiting transmission. It is not required to merge pending update messages. I.e., the dampening period applies to update record creation, not transmission.

3.11.2. Publisher capacity

It is far preferable to decline a subscription request than to accept such a request when it cannot be met.

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

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. Parentheses with a name in the middle enclose choice and case nodes. New schema objects defined here (i.e., beyond those from [I-D.draft-ietf-netconf-subscribed-notifications]) are identified with "yp".

```
module: ietf-subscribed-notifications
    +--ro streams
    |  +--ro stream* [name]
```

```
|     +--ro name                        string
|     +--ro description                 string
|     +--ro replay-support?             empty {replay}?
|     +--ro replay-log-creation-time?   yang:date-and-time {replay}?
|     +--ro replay-log-aged-time?       yang:date-and-time {replay}?
+--rw filters
|  +--rw stream-filter* [identifier]
|  |  +--rw identifier              filter-id
|  |  +--rw (filter-spec)?
|  |     +--:(stream-subtree-filter)
|  |     |  +--rw stream-subtree-filter?   <anydata> {subtree}?
|  |     +--:(stream-xpath-filter)
|  |        +--rw stream-xpath-filter?     yang:xpath1.0 {xpath}?
|  +--rw yp:selection-filter* [identifier]
|     +--rw yp:identifier                  sn:filter-id
|     +--rw (yp:filter-spec)?
|        +--:(yp:datastore-subtree-filter)
|        |  +--rw yp:datastore-subtree-filter?
|        |                                    <anydata> {sn:subtree}?
|        +--:(yp:datastore-xpath-filter)
|           +--rw yp:datastore-xpath-filter?
|                                         yang:xpath1.0 {sn:xpath}?
+--rw subscriptions
   +--rw subscription* [identifier]
      +--rw identifier                      subscription-id
      +--ro configured-subscription-state? enumeration {configured}?
      +--rw purpose?                        string {configured}?
      +--rw protocol                        transport {configured}?
      +--rw encoding                        encoding
      +--rw (target)
      |  +--:(stream)
      |  |  +--rw (stream-filter)?
      |  |  |  +--:(by-reference)
      |  |  |  |  +--rw stream-filter-ref          stream-filter-ref
      |  |  |  +--:(within-subscription)
      |  |  |     +--rw (filter-spec)?
      |  |  |        +--:(stream-subtree-filter)
      |  |  |        |  +--rw stream-subtree-filter?
      |  |  |                                      <anydata> {subtree}?
      |  |  |        +--:(stream-xpath-filter)
      |  |  |           +--rw stream-xpath-filter?
      |  |  |                                    yang:xpath1.0 {xpath}?
      |  |  +--rw stream                         stream-ref
      |  |  +--rw replay-start-time?    yang:date-and-time {replay}?
      |  +--:(yp:datastore)
      |     +--rw yp:datastore                      identityref
      |     +--rw (yp:selection-filter)?
      |        +--:(yp:by-reference)
```

```
          |       |  +--rw yp:selection-filter-ref selection-filter-ref
          |       +--:(yp:within-subscription)
          |          +--rw (yp:filter-spec)?
          |             +--:(yp:datastore-subtree-filter)
          |             |  +--rw yp:datastore-subtree-filter?
          |                                    <anydata> {sn:subtree}?
          |             +--:(yp:datastore-xpath-filter)
          |                +--rw yp:datastore-xpath-filter?
          |                                  yang:xpath1.0 {sn:xpath}?
          +--rw stop-time?                     yang:date-and-time
          +--rw dscp?                          inet:dscp {qos}?
          +--rw weighting?                     uint8 {qos}?
          +--rw dependency?                    subscription-id {qos}?
          +--rw (notification-message-origin)?
          |  +--:(interface-originated)
          |  |  +--rw source-interface?               if:interface-ref
          |  +--:(address-originated)
          |     +--rw source-vrf?                     ->
          |  /ni:network-instances/network-instance/name {supports-vrf}?
          |     +--rw source-address?            inet:ip-address-no-zone
          +--rw receivers
          |  +--rw receiver* [address port]
          |     +--rw address                  inet:host
          |     +--rw port                     inet:port-number
          |     +--ro pushed-notifications?    yang:counter64
          |     +--ro excluded-notifications?  yang:counter64
          |     +--ro status                   enumeration
          |     +---x reset
          |        +--ro output
          |           +--ro time    yang:date-and-time
          +--rw (yp:update-trigger)?
             +--:(yp:periodic)
             |  +--rw yp:periodic!
             |     +--rw yp:period         yang:timeticks
             |     +--rw yp:anchor-time?   yang:date-and-time
             +--:(yp:on-change) {on-change}?
                +--rw yp:on-change!
                   +--rw yp:dampening-period?    yang:timeticks
                   +--rw yp:no-synch-on-start?   empty
                   +--rw yp:excluded-change*     change-type

  rpcs:
    +---x establish-subscription
    |  +---w input
    |  |  +---w encoding?                    encoding
    |  |  +---w (target)
    |  |  |  +--:(stream)
    |  |  |  |  +---w (stream-filter)?
```

```
|  |  |  |  |  +--:(by-reference)
|  |  |  |  |  |  +---w stream-filter-ref         stream-filter-ref
|  |  |  |  |  +--:(within-subscription)
|  |  |  |  |     +---w (filter-spec)?
|  |  |  |  |        +--:(stream-subtree-filter)
|  |  |  |  |        |  +---w stream-subtree-filter?
|  |  |  |  |                                  <anydata> {subtree}?
|  |  |  |  |        +--:(stream-xpath-filter)
|  |  |  |  |           +---w stream-xpath-filter?
|  |  |  |  |                                yang:xpath1.0 {xpath}?
|  |  |  |  +---w stream                     stream-ref
|  |  |  |  +---w replay-start-time?    yang:date-and-time {replay}?
|  |  |  +--:(yp:datastore)
|  |  |     +---w yp:datastore                   identityref
|  |  |     +---w (yp:selection-filter)?
|  |  |        +--:(yp:by-reference)
|  |  |        |  +---w yp:selection-filter-ref selection-filter-ref
|  |  |        +--:(yp:within-subscription)
|  |  |           +---w (yp:filter-spec)?
|  |  |              +--:(yp:datastore-subtree-filter)
|  |  |              |  +---w yp:datastore-subtree-filter?
|  |  |              |                      <anydata> {sn:subtree}?
|  |  |              +--:(yp:datastore-xpath-filter)
|  |  |                 +---w yp:datastore-xpath-filter?
|  |  |                                    yang:xpath1.0 {sn:xpath}?
|  |  +---w stop-time?                     yang:date-and-time
|  |  +---w dscp?                          inet:dscp {qos}?
|  |  +---w weighting?                     uint8 {qos}?
|  |  +---w dependency?                    subscription-id {qos}?
|  |  +---w (yp:update-trigger)?
|  |     +--:(yp:periodic)
|  |     |  +---w yp:periodic!
|  |     |     +---w yp:period         yang:timeticks
|  |     |     +---w yp:anchor-time?   yang:date-and-time
|  |     +--:(yp:on-change) {on-change}?
|  |        +---w yp:on-change!
|  |           +---w yp:dampening-period?    yang:timeticks
|  |           +---w yp:no-synch-on-start?   empty
|  |           +---w yp:excluded-change*     change-type
|  +--ro output
|     +--ro identifier    subscription-id
+---x modify-subscription
|  +---w input
|     +---w identifier?                   subscription-id
|     +---w (target)
|     |  +--:(stream)
|     |  |  +---w (stream-filter)?
|     |  |     +--:(by-reference)
```

```
   |     |  |     |  +---w stream-filter-ref         stream-filter-ref
   |     |  |     +--:(within-subscription)
   |     |  |        +---w (filter-spec)?
   |     |  |           +--:(stream-subtree-filter)
   |     |  |           |  +---w stream-subtree-filter?
   |     |  |                                    <anydata> {subtree}?
   |     |  |           +--:(stream-xpath-filter)
   |     |  |              +---w stream-xpath-filter?
   |     |  |                                     yang:xpath1.0 {xpath}?
   |     |  +--:(yp:datastore)
   |     |     +---w (yp:selection-filter)?
   |     |        +--:(yp:by-reference)
   |     |        |  +---w yp:selection-filter-ref selection-filter-ref
   |     |        +--:(yp:within-subscription)
   |     |           +---w (yp:filter-spec)?
   |     |              +--:(yp:datastore-subtree-filter)
   |     |              |  +---w yp:datastore-subtree-filter?
   |     |              |                       <anydata> {sn:subtree}?
   |     |              +--:(yp:datastore-xpath-filter)
   |     |                 +---w yp:datastore-xpath-filter?
   |     |                                     yang:xpath1.0 {sn:xpath}?
   |     +---w stop-time?                      yang:date-and-time
   |     +---w (yp:update-trigger)?
   |        +--:(yp:periodic)
   |        |  +---w yp:periodic!
   |        |     +---w yp:period         yang:timeticks
   |        |     +---w yp:anchor-time?   yang:date-and-time
   |        +--:(yp:on-change) {on-change}?
   |           +---w yp:on-change!
   |              +---w yp:dampening-period?   yang:timeticks
   +---x delete-subscription
   |  +---w input
   |     +---w identifier    subscription-id
   +---x kill-subscription
      +---w input
         +---w identifier    subscription-id

 yang-data (for placement into rpc error responses)
   +-- establish-subscription-error-stream
   |  +--ro reason?                   identityref
   |  +--ro filter-failure-hint?      string
   |  +--ro replay-start-time-hint?   yang:date-and-time
   +-- modify-subscription-error-stream
      +--ro reason?                identityref
      +--ro filter-failure-hint?   string

 notifications:
   +---n replay-completed {replay}?
```

```
   |  +--ro identifier    subscription-id
   +---n subscription-completed
   |  +--ro identifier    subscription-id
   +---n subscription-started {configured}?
   |  +--ro identifier                    subscription-id
   |  +--ro protocol                      transport {configured}?
   |  +--ro encoding                      encoding
   |  +--ro (target)
   |  |  +--:(stream)
   |  |  |  +--ro (stream-filter)?
   |  |  |  |  +--:(by-reference)
   |  |  |  |  |  +--ro stream-filter-ref            stream-filter-ref
   |  |  |  |  +--:(within-subscription)
   |  |  |  |     +--ro (filter-spec)?
   |  |  |  |        +--:(stream-subtree-filter)
   |  |  |  |        |  +--ro stream-subtree-filter?
   |  |  |  |        |                              <anydata> {subtree}?
   |  |  |  |        +--:(stream-xpath-filter)
   |  |  |  |           +--ro stream-xpath-filter?
   |  |  |  |                                       yang:xpath1.0 {xpath}?
   |  |  |  +--ro stream                           stream-ref
   |  |  |  +--ro replay-start-time?       yang:date-and-time {replay}?
   |  |  +--:(yp:datastore)
   |  |     +--ro yp:datastore                     identityref
   |  |     +--ro (yp:selection-filter)?
   |  |        +--:(yp:by-reference)
   |  |        |  +--ro yp:selection-filter-ref   selection-filter-ref
   |  |        +--:(yp:within-subscription)
   |  |           +--ro (yp:filter-spec)?
   |  |              +--:(yp:datastore-subtree-filter)
   |  |              |  +--ro yp:datastore-subtree-filter?
   |  |                                        <anydata> {sn:subtree}?
   |  |              +--:(yp:datastore-xpath-filter)
   |  |                 +--ro yp:datastore-xpath-filter?
   |  |                                       yang:xpath1.0 {sn:xpath}?
   |  +--ro stop-time?                       yang:date-and-time
   |  +--ro dscp?                            inet:dscp {qos}?
   |  +--ro weighting?                       uint8 {qos}?
   |  +--ro dependency?                      subscription-id {qos}?
   |  +--ro (yp:update-trigger)?
   |     +--:(yp:periodic)
   |     |  +--ro yp:periodic!
   |     |     +--ro yp:period         yang:timeticks
   |     |     +--ro yp:anchor-time?   yang:date-and-time
   |     +--:(yp:on-change) {on-change}?
   |        +--ro yp:on-change!
   |           +--ro yp:dampening-period?    yang:timeticks
   |           +--ro yp:no-synch-on-start?   empty
```

```
|          +--ro yp:excluded-change*     change-type
+---n subscription-resumed
|  +--ro identifier    subscription-id
+---n subscription-modified
|  +--ro identifier                     subscription-id
|  +--ro protocol                       transport {configured}?
|  +--ro encoding                       encoding
|  +--ro (target)
|  |  +--:(stream)
|  |  |  +--ro (stream-filter)?
|  |  |  |  +--:(by-reference)
|  |  |  |  |  +--ro stream-filter-ref            stream-filter-ref
|  |  |  |  +--:(within-subscription)
|  |  |  |     +--ro (filter-spec)?
|  |  |  |        +--:(stream-subtree-filter)
|  |  |  |        |  +--ro stream-subtree-filter?
|  |  |  |        |                               <anydata> {subtree}?
|  |  |  |        +--:(stream-xpath-filter)
|  |  |  |           +--ro stream-xpath-filter?
|  |  |  |                                        yang:xpath1.0 {xpath}?
|  |  |  +--ro stream                             stream-ref
|  |  |  +--ro replay-start-time?       yang:date-and-time {replay}?
|  |  +--:(yp:datastore)
|  |     +--ro yp:datastore                       identityref
|  |     +--ro (yp:selection-filter)?
|  |        +--:(yp:by-reference)
|  |        |  +--ro yp:selection-filter-ref    selection-filter-ref
|  |        +--:(yp:within-subscription)
|  |           +--ro (yp:filter-spec)?
|  |              +--:(yp:datastore-subtree-filter)
|  |              |  +--ro yp:datastore-subtree-filter?
|  |              |                           <anydata> {sn:subtree}?
|  |              +--:(yp:datastore-xpath-filter)
|  |                 +--ro yp:datastore-xpath-filter?
|  |                                        yang:xpath1.0 {sn:xpath}?
|  +--ro stop-time?                        yang:date-and-time
|  +--ro dscp?                             inet:dscp {qos}?
|  +--ro weighting?                        uint8 {qos}?
|  +--ro dependency?                       subscription-id {qos}?
|  +--ro (yp:update-trigger)?
|     +--:(yp:periodic)
|     |  +--ro yp:periodic!
|     |     +--ro yp:period         yang:timeticks
|     |     +--ro yp:anchor-time?   yang:date-and-time
|     +--:(yp:on-change) {on-change}?
|        +--ro yp:on-change!
|           +--ro yp:dampening-period?    yang:timeticks
|           +--ro yp:no-synch-on-start?   empty
```

```
    |           +--ro yp:excluded-change*     change-type
    +---n subscription-terminated
    |  +--ro identifier    subscription-id
    |  +--ro reason        identityref
    +---n subscription-suspended
       +--ro identifier    subscription-id
       +--ro reason        identityref

module: ietf-yang-push

  rpcs:
    +---x resynch-subscription {on-change}?
       +---w input
          +---w identifier    sn:subscription-id

  yang-data: (for placement into rpc error responses)
    +-- resynch-subscription-error
    |  +--ro reason?                   identityref
    |  +--ro period-hint?              timeticks
    |  +--ro filter-failure-hint?      string
    |  +--ro object-count-estimate?    uint32
    |  +--ro object-count-limit?       uint32
    |  +--ro kilobytes-estimate?       uint32
    |  +--ro kilobytes-limit?          uint32
    +-- establish-subscription-error-datastore
    |  +--ro reason?                   identityref
    |  +--ro period-hint?              timeticks
    |  +--ro filter-failure-hint?      string
    |  +--ro object-count-estimate?    uint32
    |  +--ro object-count-limit?       uint32
    |  +--ro kilobytes-estimate?       uint32
    |  +--ro kilobytes-limit?          uint32
    +-- modify-subscription-error-datastore
       +--ro reason?                   identityref
       +--ro period-hint?              timeticks
       +--ro filter-failure-hint?      string
       +--ro object-count-estimate?    uint32
       +--ro object-count-limit?       uint32
       +--ro kilobytes-estimate?       uint32
       +--ro kilobytes-limit?          uint32

  notifications:
    +---n push-update
    |  +--ro subscription-id?      sn:subscription-id
    |  +--ro incomplete-update?     empty
    |  +--ro datastore-contents?   <anydata>
    +---n push-change-update {on-change}?
       +--ro subscription-id?     sn:subscription-id
```

```
      +--ro incomplete-update?    empty
      +--ro datastore-changes?   <anydata>
```

Figure 6: Model structure

Selected components of the model are summarized below.

4.2. Subscription configuration

Both configured and dynamic subscriptions are represented within the list subscription. New and enhanced parameters extending the basic subscription data model in [I-D.draft-ietf-netconf-subscribed-notifications] include:

- The targeted datastore from which the selection is being made. The potential datastores include those from [I-D.draft-ietf-netmod-revised-datastores]. A platform may also choose to support a custom datastore.

- A selection filter identifying yang nodes of interest within a datastore. Filter contents are specified via a reference to an existing filter, or via an in-line definition for only that subscription. Referenced filters allows an implementation to avoid evaluating filter acceptability during a dynamic subscription request. The case statement differentiates the options.

- For periodic subscriptions, triggered updates will occur at the boundaries of a specified time interval. These boundaries many be calculated from the periodic parameters:

  * 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" from an "anchor-time". If "anchor-time" is not provided, then the "anchor-time" MUST be set with the creation time of the initial update record.

- For on-change subscriptions, assuming any dampening period has completed, triggering 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. If no dampening period is in effect, 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.

* an "excluded-change" flag which allows restriction of the types of changes for which updates should be sent (e.g., only add to an update record on object creation).

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

4.3. YANG Notifications

4.3.1. State Change Notifications

Subscription state notifications and mechanism are reused from [I-D.draft-ietf-netconf-subscribed-notifications]. Some have been augmented to include the datastore specific objects.

4.3.2. Notifications for Subscribed Content

Along with the subscribed content, there are other objects which might be part of a "push-update" or "push-change-update"

A "subscription-id" MUST be transported along with the subscribed contents. An [RFC5277] Section 4 one-way notification MAY be used for encoding updates. Where it is, the relevant "subscription-id" MUST be encoded as the first element within each "push-update" or "push-change-update". This allows a receiver to differentiate which subscription resulted in a particular push.

A "time-of-update" which represents the time an update record snapshot was generated. A receiver MAY assume that a publisher's objects have these pushed values at this point in time.

An "incomplete-update" object. This object 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 a datastore was unable to providing the full set of datastore nodes to a publisher process.) To facilitate re-synchronization of on-change subscriptions, a publisher MAY subsequently send a "push-update" containing a full selection snapshot of subscribed data.

4.4. YANG RPCs

YANG-Push subscriptions are established, modified, and deleted using RPCs augmented from [I-D.draft-ietf-netconf-subscribed-notifications].

4.4.1. 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-subscribed-notifications"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore>
      <yp:source xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
        ds:operational
      </yp:source>
      <xpath-filter
          xmlns:ex="http://example.com/sample-data/1.0"
          select="/ex:foo"/>
    </yp:datastore>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
 </establish-subscription>
</netconf:rpc>
```

Figure 7: 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 "identifier" 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-subscribed-notifications">
       ok
    </subscription-result>
    <identifier
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
       52
    </identifier>
</rpc-reply>
```

Figure 8: Establish-subscription positive RPC response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, no capacity to serve the subscription at the publisher, or the inability of the publisher to select datastore content at the requested cadence.

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error hints which help a subscriber understand subscription parameters might have been accepted for the request. These hints would be included within the yang-data structure "establish-subscription-error-datastore". However even with these hints, there are no guarantee that subsequent requests will in fact be accepted.

The specific parameters to be returned in as part of the RPC error response depend on the specific transport that is used to manage the subscription. In the case of NETCONF [I-D.draft-ietf-netconf-netconf-event-notifications], when a subscription request is rejected, the NETCONF RPC reply MUST include an "rpc-error" element with the following elements:

- "error-type" of "application".
- "error-tag" of "operation-failed".
- Optionally, an "error-severity" of "error" (this MAY but does not have to be included).
- "error-app-tag" with the value being a string that corresponds to an identity with a base of "establish-subscription-error".
- Optionally, "error-info" containing XML-encoded data with hints for parameter settings that might result in future RPC success per yang-data definition "establish-subscription-error-datastore".

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-subscribed-notifications"
  xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter netconf:type="xpath"
    xmlns:ex="http://example.com/sample-data/1.0">
      /ex:foo
    </yp:datastore-xpath-filter>
    <yp:on-change>
      <yp:dampening-period>100</yp:dampening-period>
    </yp:on-change>
  </establish-subscription>
</netconf:rpc>
```

Figure 9: 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">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>
        on-change-unsupported
    </error-message>
    <error-path
   xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
      /yp:periodic/yp:period
    </error-path>
  </rpc-error>
</rpc-reply>
```

Figure 10: Establish-subscription error response example 2

4.4.2. Modify-subscription RPC

The subscriber MAY invoke the "modify-subscription" RPC for a subscription it previously established. The subscriber will include newly desired values in the "modify-subscription" RPC. Parameters not included MUST remain unmodified. Below is an example where a subscriber attempts to modify the "period" of a 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-subscribed-notifications"
   xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <identifier>1011</identifier>
    <yp:datastore
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
    netconf:type="xpath" xmlns:ex="http://example.com/sample-data/1.0">
      /ex:bar
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>250</yp:period>
    </yp:periodic>
   </modify-subscription>
</netconf:rpc>
```

Figure 11: Modify subscription request

The publisher MUST respond explicitly positively or negatively to the request. If the subscription modification is rejected, the subscription is maintained as it was before the modification request. In addition, the publisher MUST send an rpc error response. This rpc error response may contain hints encapsulated within the yang-data structure "modify-subscription-error-datastore". A subscription MAY be modified multiple times.

The specific parameters to be returned in as part of the RPC error response depend on the specific transport that is used to manage the subscription. In the case of NETCONF [I-D.draft-ietf-netconf-netconf-event-notifications], when a subscription request is rejected, the NETCONF RPC reply MUST include an "rpc-error" element with the following elements:

- "error-type" of "application".

- "error-tag" of "operation-failed".

- Optionally, an "error-severity" of "error" (this MAY but does not have to be included).

- "error-app-tag" with the value being a string that corresponds to an identity with a base of "modify-subscription-error".

- "error-path" pointing to the object or parameter that caused the rejection.

- Optionally, "error-info" containing XML-encoded data with hints for parameter settings that might result in future RPC success per yang-data definition "modify-subscription-error-datastore".

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

4.4.3. 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's "identifier". This RPC is unmodified from [I-D.draft-ietf-netconf-subscribed-notifications].

4.4.4. Resynch-subscription RPC

This RPC is only applicable only for on-change subscriptions previously established using an "establish-subscription" RPC. For example:

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

Resynch subscription

On receipt, a publisher must either accept the request and quickly follow with a "push-update", or send an appropriate error within an rpc error response. Within an error response, the publisher may include supplemental information about the reasons within the yang-data structure "resynch-subscription-error".

4.4.5. YANG Module Synchronization

To make subscription requests, the subscriber needs to know the YANG module library available on the publisher. 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]. This YANG library information is important for the receiver to reproduce the set of object definitions used within the publisher.

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 publisher implementation). In this case, the receiver needs to be informed of module changes before datastore nodes from changed modules can be processed correctly. The YANG library provides a simple "yang-library-change" notification that informs the subscriber that the library has changed. The receiver then needs to re-read the entire YANG library data for the replicated publisher 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@2018-02-23.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-subscribed-notifications {
    prefix sn;
  }
  import ietf-datastores {
```

```
    prefix ds;
  }
  import ietf-restconf   {
    prefix rc;
  }

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

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

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

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

     Editor:   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 YANG specifications for YANG push.";

  revision 2018-02-23 {
    description
      "Initial revision.";
    reference
      "draft-ietf-netconf-yang-push-15";
  }

 /*
  * FEATURES
  */

  feature on-change {
    description
```

```
      "This feature indicates that on-change triggered subscriptions
       are supported.";
  }

 /*
  * IDENTITIES
  */

  /* Error type identities for datastore subscription  */

  identity resynch-subscription-error {
     description
      "Problem found while attempting to fulfill an
      'resynch-subscription' RPC request. ";
  }

  identity cant-exclude {
    base sn:establish-subscription-error;
    description
      "Unable to remove the set of 'excluded-changes'.  This means the
      publisher is unable to restrict 'push-change-update's to just the
      change types requested for this subscription.";
  }

  identity datastore-not-subscribable {
    base sn:establish-subscription-error;
    base sn:subscription-terminated-reason;
    description
      "This is not a subscribable datastore.";
  }

  identity no-such-subscription-resynch {
    base resynch-subscription-error;
    description
     "Referenced subscription doesn't exist. This may be as a result of
      a non-existent subscription ID, an ID which belongs to another
      subscriber, or an ID for configured subscription.";
  }

  identity on-change-unsupported {
    base sn:establish-subscription-error;
    description
      "On-change is not supported for any objects which are selectable
      by this filter.";
  }

  identity on-change-synch-unsupported {
    base sn:establish-subscription-error;
```

```
    description
      "Neither synch on start nor resynchronization are supported for
      this subscription.  This error will be used for two reasons. First
      if an 'establish-subscription' RPC doesn't include
      'no-synch-on-start', yet the publisher can't support sending a
      'push-update' for this subscription for reasons other than
      'on-change-unsupported' or 'synchronization-size'.  And second, if
      the 'resynch-subscription' RPC is invoked either for an existing
      periodic subscription, or for an on-change subscription which
      can't support resynchronization.";
  }

  identity period-unsupported {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base sn:subscription-suspended-reason;
    description
      "Requested time period is too short. This can be for both
       periodic and on-change subscriptions (with or without
       dampening.)

      Hints suggesting alternative periods may be returned as
      supplemental information when this expressed.";
  }

  identity result-too-big {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base sn:subscription-suspended-reason;
    description
      "Periodic or on-change push update datatrees exceed a maximum size
      limit.  Hints on estimated size of what was too big may be
      returned as supplemental information when this expressed.";
  }

  identity synchronization-size {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base resynch-subscription-error;
    base sn:subscription-suspended-reason;
    description
      "Synch-on-start or resynchronization datatree exceeds a maximum
      size limit.

      Where this identity is referenced as an 'error-app-tag' within an
      RPC response, the response's 'error-info' may contain:";
  }
```

```
  identity unchanging-selection {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base sn:subscription-terminated-reason;
    description
      "Selection filter is unlikely to ever select datatree nodes.  This
      means that based on the subscriber's current access rights, the
      publisher recognizes that the selection filter is unlikely to ever
      select datatree nodes which change. Examples for this might be
      that node or subtree doesn't exist, read access is not permitted
      for a receiver, or static objects that only change at reboot have
      been chosen.";
  }

  /*
   * TYPE DEFINITIONS
   */

  typedef change-type {
    type enumeration {
      enum "create" {
        description
          "Create a new data resource if it does not already exist.  If
          it already exists, replace it.";
      }
      enum "delete" {
        description
          "Delete a data resource if it already exists. If it does not
          exist, take no action.";
      }
      enum "insert" {
        description
          "Insert a new user-ordered data resource";
      }
      enum "merge" {
        description
          "merge the edit value with the target data resource; create
          if it does not already exist";
      }
      enum "move" {
        description
          "Reorder the target data resource";
      }
      enum "replace" {
        description
          "Replace the target data resource with the edit value";
      }
      enum "remove" {
```

```
        description
          "Remove a data resource if it already exists ";
      }
    }
    description
      "Specifies different types of datastore changes.";
    reference
      "RFC 8072 section 2.5, with a delta that it is valid for a
      receiver to process an update record which performs a create
      operation on a datastore node the receiver believes exists, or to
      process a delete on a datastore node the receiver believes is
      missing.";
  }

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

  /*
   * GROUP DEFINITIONS
   */

  grouping datastore-criteria {
    description
      "A grouping to define criteria for which selected objects from
       a targeted datastore should be included in push updates.";
    leaf datastore {
       type identityref {
         base ds:datastore;
       }
       mandatory true;
       description
         "Datastore from which to retrieve data.";
    }
    uses selection-filter-objects;
  }

  grouping selection-filter-types {
    description
      "This grouping defines the types of selectors for objects from a
       datastore.";
    choice filter-spec {
      description
        "The content filter specification for this request.";
```

```
      anydata datastore-subtree-filter {
        if-feature "sn:subtree";
        description
          "This parameter identifies the portions of the
           target datastore to retrieve.";
      }
      leaf datastore-xpath-filter {
        if-feature "sn:xpath";
        type yang:xpath1.0;
        description
          "This parameter contains an XPath expression identifying the
          portions of the target datastore to retrieve.

          If the expression returns a node-set, all nodes in the
          node-set are selected by the filter.  Otherwise, if the
          expression does not return a node-set, the filter
          doesn't select any nodes.

          The expression is evaluated in the following XPath context:

           o  The set of namespace declarations are those in scope on
              the 'xpath-filter' leaf element.

           o  The set of variable bindings is empty.

           o  The function library is the core function library, and
              the XPath functions defined in section 10 in RFC 7950.

           o  The context node is the root node of the target
              datastore.";
      }
    }
  }

  grouping selection-filter-objects {
    description
      "This grouping defines a selector for objects from a
       datastore.";
    choice selection-filter {
      description
        "The source of the selection filter applied to the subscription.
        This will come either referenced from a global list, or be
        provided within the subscription itself.";
      case by-reference {
        description
          "Incorporate a filter that has been configured separately.";
        leaf selection-filter-ref {
          type selection-filter-ref;
```

```
          mandatory true;
          description
            "References an existing selection filter which is to be
             applied to the subscription.";
        }
      }
      case within-subscription {
        description
          "Local definition allows a filter to have the same lifecycle
          as the subscription.";
        uses selection-filter-types;

      }
    }
  }

  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 record to
         the subscriber.";
      case periodic {
        container periodic {
          presence "indicates an periodic subscription";
          description
            "The publisher is requested to notify periodically the
            current values of the datastore as defined by the selection
            filter.";
          leaf period {
            type yang:timeticks;
            mandatory true;
            description
              "Duration of time which should occur between periodic
               push updates.";
          }
          leaf anchor-time {
            type yang:date-and-time;
            description
              "Designates a timestamp before or after which a series of
               periodic push updates are determined. The next update
               will take place at a whole multiple interval from the
               anchor time.  For example, for an anchor time is set for
               the top of a particular minute and a period interval of a
               minute, updates will be sent at the top of every minute
```

```
               this subscription is active.";
          }
        }
      }
      case on-change {
        if-feature "on-change";
        container on-change {
          presence "indicates an on-change subscription";
        description
            "The publisher is requested to notify changes in values in
             the datastore subset as defined by a selection filter.";
          leaf dampening-period {
            type yang:timeticks;
            default 0;
            description
              "Specifies the minimum interval between the assembly of
                successive update records for a single receiver of a
              subscription. Whenever subscribed objects change, and a
              dampening period interval (which may be zero) has elapsed
              since the previous update record creation for a receiver,
              then any subscribed objects and properties which have
              changed since the previous update record will have their
              current values marshalled and placed into a new update
              record.";
          }
        }
      }
    }
  }

  grouping update-policy {
    description
      "This grouping describes the datastore specific subscription
       conditions of a subscription.";
    uses update-policy-modifiable {
      augment "update-trigger/on-change/on-change" {
        description
          "Includes objects not modifiable once subscription is
           established.";
        leaf no-synch-on-start {
          type empty;
          description
            "The presence of this object restricts an on-change
             subscription from sending push-update notifications.  When
             present, pushing a full selection per the terms of the
             selection filter MUST NOT be done for this subscription.
             Only updates about changes, 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.  After that, push-change-update
            notifications are exclusively sent unless the publisher
            chooses to resynch the subscription via a new push-update
            notification.";
        }
        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 hints {
    description
      "Parameters associated with some error for a subscription made
      upon a datastore.";
    leaf period-hint {
      type yang:timeticks;
      description
        "Returned when the requested time period is too short. This
         hint can assert a viable period for either a periodic push
         cadence or an on-change dampening interval.";
    }
    leaf filter-failure-hint {
      type string;
        description
          "Information describing where and/or why a provided filter
           was unsupportable for a subscription.";
    }
    leaf object-count-estimate {
      type uint32;
      description
        "If there are too many objects which could potentially be
         returned by the selection 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
         selection 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 selection 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.";
    }
  }


  /*
   * RPCs
   */

   rpc resynch-subscription {
    if-feature "on-change";
    description
      "This RPC allows a subscriber of an active on-change
       subscription to request a full push of objects in their current
       state. A successful result would invoke a push-update of all
       datastore objects that the subscriber is permitted to access.
       This request may only come from the same subscriber using the
       establish-subscription RPC.";
    input {
      leaf identifier {
        type sn:subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be resynched.";
      }
    }
  }

  rc:yang-data resynch-subscription-error {
    container resynch-subscription-error {
      description
        "If a 'resynch-subscription' RPC fails, the subscription is not
        resynched and the RPC error response MUST indicate the reason
        for this failure. This yang-data MAY be inserted as structured
```

```
        data within a subscription's RPC error response to indicate the
        failure reason.";
      leaf reason {
        type identityref {
          base resynch-subscription-error;
        }
        mandatory true;
        description
          "Indicates the reason why the publisher has declined a request
          for subscription resynchronization.";
      }
      uses hints;
    }
  }

  augment "/sn:establish-subscription/sn:input" {
    description
      "This augmentation adds additional subscription parameters that
      apply specifically to datastore updates to RPC input.";
    uses update-policy;
  }

  augment "/sn:establish-subscription/sn:input/sn:target" {
    description
      "This augmentation adds the datastore as a valid target
      for the subscription to RPC input.";
    case datastore {
      description
        "Information specifying the parameters of an request for a
        datastore subscription.";
      uses datastore-criteria;
    }
  }

  rc:yang-data establish-subscription-error-datastore {
    container establish-subscription-error-datastore {
      description
        "If any 'establish-subscription' RPC parameters are
        unsupportable against the datastore, a subscription is not
        created and the RPC error response MUST indicate the reason why
        the subscription failed to be created. This yang-data MAY be
        inserted as structured data within a subscription's RPC error
        response to indicate the failure reason.  This yang-data MUST be
        inserted if hints are to be provided back to the subscriber.";
      leaf reason {
        type identityref {
          base sn:establish-subscription-error;
        }
```

```
        description
          "Indicates the reason why the subscription has failed to
          be created to a targeted datastore.";
      }
      uses hints;
    }
  }

  augment "/sn:modify-subscription/sn:input" {
    description
      "This augmentation adds additional subscription parameters
       specific to datastore updates.";
    uses update-policy-modifiable;
  }

  augment "/sn:modify-subscription/sn:input/sn:target" {
    description
      "This augmentation adds the datastore as a valid target
      for the subscription to RPC input.";
    case datastore {
      description
        "Information specifying the parameters of an request for a
         datastore subscription.";
      uses selection-filter-objects;
    }
  }

  rc:yang-data modify-subscription-error-datastore {
    container modify-subscription-error-datastore {
      description
        "This yang-data MAY be provided as part of a subscription's RPC
        error response when there is a failure of a
        'modify-subscription' RPC which has been made against a
        datastore.  This yang-data MUST be used if hints are to be
        provides back to the subscriber.";
      leaf reason {
        type identityref {
          base sn:modify-subscription-error;
        }
        description
          "Indicates the reason why the subscription has failed to
          be modified.";
      }
      uses hints;
    }
  }

  /*
```

```
   * NOTIFICATIONS
   */

  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;
      description
        "This references the subscription which drove the notification
         to be sent.";
    }
    leaf incomplete-update {
      type empty;
      description
        "This is a flag which indicates that not all datastore nodes
         subscribed to are included with this update. In other words,
         the publisher has failed to fulfill its full subscription
         obligations, and despite its best efforts is providing an
         incomplete set of objects.";
    }
    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
         selection 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;
      description
```

```
        "This references the subscription which drove the notification
         to be sent.";
    }
    leaf incomplete-update {
      type empty;
      description
        "The presence of this object indicates 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.";
    }
    anydata datastore-changes {
      description
        "This contains the set of datastore changes needed
         to update a remote datastore starting at the time of 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
         datastore implied by the previous update to result in the
         current state.";
      reference
        "RFC 8072 section 2.5, with a delta that it is ok to receive
         ability create on an existing node, or receive a delete on a
         missing node.";
    }
  }

  augment "/sn:subscription-started" {
    description
      "This augmentation adds many datastore specific objects to
       the notification that a subscription has started.";
    uses update-policy;
  }

  augment "/sn:subscription-started/sn:target" {
    description
      "This augmentation allows the datastore to be included as part
       of the notification that a subscription has started.";
    case datastore {
       uses datastore-criteria {
          refine "selection-filter/within-subscription" {
          description
            "Specifies where the selection filter, and where it came
            from within the subscription and then populated within this
            notification. If the 'selection-filter-ref' is populated,
            the filter within the subscription came from the 'filters'
            container.  Otherwise it is populated in-line as part of the
```

```
            subscription itself.";
        }
      }
    }
  }

  augment "/sn:subscription-modified" {
    description
      "This augmentation adds many datastore specific objects to
       the notification that a subscription has been modified.";
    uses update-policy;
  }
  augment "/sn:subscription-modified/sn:target" {
    description
      "This augmentation allows the datastore to be included as part
      of the notification that a subscription has been modified.";
    case datastore {
       uses datastore-criteria {
          refine "selection-filter/within-subscription" {
          description
            "Specifies where the selection filter, and where it came
            from within the subscription and then populated within this
            notification. If the 'selection-filter-ref' is populated,
            the filter within the subscription came from the 'filters'
            container.  Otherwise it is populated in-line as part of the
            subscription itself.";
        }
      }
    }
  }

  /*
   * DATA NODES
   */

  augment "/sn:filters" {
    description
      "This augmentation allows the datastore to be included as part
       of the selection filtering criteria for a subscription.";
    list selection-filter {
      key "identifier";
      description
        "A list of pre-positioned filters that can be applied
         to datastore subscriptions.";
      leaf identifier {
        type sn:filter-id;
        description
          "An identifier to differentiate between selection filters.";
```

```
      }
      uses selection-filter-types;
    }
  }

  augment "/sn:subscriptions/sn:subscription" {
    description
      "This augmentation adds many datastore specific objects to a
       subscription.";
    uses update-policy;
  }
  augment "/sn:subscriptions/sn:subscription/sn:target" {
    description
      "This augmentation allows the datastore to be included as part
      of the selection filtering criteria for a subscription.";
    case datastore {
       uses datastore-criteria;
    }
  }
}

<CODE ENDS>
```

6. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-yang-push
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

Name: ietf-yang-push
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-push
Prefix: yp
Reference: draft-ietf-netconf-yang-push-12.txt (RFC form)

7. Security Considerations

All security considerations from [I-D.draft-ietf-netconf-subscribed-notifications] are relevant for datastores. In addition there are specific security considerations for receivers defined in Section 3.9

If the access control permissions on subscribed YANG nodes change during the lifecycle of a subscription, a publisher MUST either transparently conform to the new access control permissions, or must terminate or restart the subscriptions so that new access control permissions are re-established.

The NETCONF Authorization Control Model SHOULD be used to restrict the delivery of YANG nodes for which the receiver has no access.

8. Acknowledgments

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

9. References

9.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-06 (work in progress), January 2018.

[I-D.draft-ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-04 (work in progress), August 2017.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

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

[RFC6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-05 (work in progress), September 2017.

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

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

9.2. Informative References

[I-D.draft-ietf-netconf-netconf-event-notifications] Gonzalez Prieto, A., Clemm, A., Voit, E., Nilsen-Nygaard, E., and A. Tripathy, "NETCONF Support for Event Notifications", September 2017.

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

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

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

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

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

Appendix A. Appendix A: Subscription Errors

A.1. RPC Failures

Rejection of an RPC for any reason is indicated by via RPC error response from the publisher. Valid RPC errors returned include both existing transport layer RPC error codes, such as those seen with NETCONF in [RFC6241], as well as subscription specific errors such as those defined within the YANG model. As a result, how subscription errors are encoded within an RPC error response is transport dependent.

References to specific identities within the either the subscribed-notifications YANG model or the yang-push YANG model may be returned as part of the error responses resulting from failed attempts at datastore subscription. Following are valid errors per RPC (note: throughout this section the prefix 'sn' indicates an item imported from the subscribed-notifications.yang model):

establish-subscription	modify-subscription
cant-exclude	sn:filter-unsupported
datastore-not-subscribable	sn:insufficient-resources
sn:dscp-unavailable	sn:no-such-subscription
sn:filter-unsupported	period-unsupported
sn:insufficient-resources	result-too-big
on-change-unsupported	synchronization-size
on-change-synch-unsupported	unchanging-selection
period-unsupported	
result-too-big	resynch-subscription
synchronization-size	
unchanging-selection	no-such-subscription-resynch
	synchronization-size

delete-subscription	kill-subscription
sn:no-such-subscription	sn:no-such-subscription

There is one final set of transport independent RPC error elements included in the YANG model. These are the following four yang-data structures for failed datastore subscriptions:

1. yang-data establish-subscription-error-datastore
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "establish-subscription" RPC response. This MUST be sent if hints are included.

2. yang-data modify-subscription-error-datastore
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "modifiy-subscription" RPC response. This MUST be sent if hints are included.

3. yang-data sn:delete-subscription-error
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.

4. yang-data resynch-subscription-error
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "resynch-subscription" RPC response.

A.2. Notifications of Failure

A subscription may be unexpectedly terminated or suspended independent of any RPC or configuration operation. In such cases, indications of such a failure MUST be provided. To accomplish this, the following types of error identities may be returned within the corresponding subscription state change notification:

subscription-terminated	subscription-suspended
datastore-not-subscribable	sn:insufficient-resources
sn:filter-unavailable	period-unsupported
sn:no-such-subscription	result-too-big
sn:suspension-timeout	synchronization-size
unchanging-selection	

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v14 - v15

- Minor text fixes. Includes a fix to on-change update calculation to cover churn when an object changes to and from a value during a dampening period.

v13 - v14

- Minor text fixes.

v12 - v13

- Hint negotiation models now show error examples.
- yang-data structures for rpc errors.

v11 - v12

- Included Martin's review clarifications.
- QoS moved to subscribed-notifications
- time-of-update removed as it is redundant with RFC5277's eventTime, and other times from notification-messages.
- Error model moved to match existing implementations
- On-change notifiable removed, how to do this is implementation specific.
- NMDA model supported. Non NMDA version at https://github.com/netconf-wg/yang-push/

v10 - v11

- Promise model reference added.
- Error added for no-such-datastore
- Inherited changes from subscribed notifications (such as optional feature definitions).
- scrubbed the examples for proper encodings

v09 - v10

- Returned to the explicit filter subtyping of v00-v05
- identityref to ds:datastore made explicit

- Returned ability to modify a selection filter via RPC.

v08 - v09

- Minor tweaks cleaning up text, removing appendicies, and making reference to revised-datastores.
- Subscription-id optional in push updates, except when encoded in RFC5277, Section 4 one-way notification.
- Finished adding the text descibing the resynch subscription RPC.
- Removed relationships to other drafts and future technology appendicies as this work is being explored elsewhere.
- Deferred the multi-line card issue to new drafts
- Simplified the NACM interactions.

v07 - v08

- Updated YANG models with minor tweaks to accommodate changes of ietf-subscribed-notifications.

v06 - v07

- Clarifying text tweaks.
- Clarification that filters act as selectors for subscribed datastore nodes; support for value filters not included but possible as a future extension
- Filters don't have to be matched to existing YANG objects

v05 - v06

- Security considerations updated.
- Base YANG model in [subscribe] updated as part of move to identities, YANG augmentations in this doc matched up
- Terms refined and text updates throughout
- Appendix talking about relationship to other drafts added.
- Datastore replaces stream
- Definitions of filters improved

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
VMware

Email: agonzalezpri@vmware.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