NETCONF   A. Clemm
Internet-Draft   Huawei
Intended status: Standards Track   E. Voit
Expires: April 25, 2019   Cisco Systems
                                                    A. Gonzalez Prieto
                                                                 VMware
                                                            A. Tripathy
                                                    E. Nilsen-Nygaard
                                                         Cisco Systems
                                                            A. Bierman
                                                             YumaWorks
                                                            B. Lengyel
                                                              Ericsson
                                                      October 22, 2018

                    Subscription to YANG Datastores
                    draft-ietf-netconf-yang-push-20

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 updates enables new
   capabilities based on the remote mirroring and monitoring of
   configuration and operational state.

Copyright Notice

Table of Contents

1. Introduction

   Traditional approaches to providing visibility into managed entities
   from remote 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:

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

   o 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.

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

   o For applications that monitor for changes, many remote polling
      cycles place unwanted and ultimately wasteful load on the network,
      devices, and applications, particularly when changes occur only
      infrequently.

   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 server (also referred to as 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

   This document uses the terminology defined in [RFC7950], [RFC8341],
   [RFC8342], and [I-D.draft-ietf-netconf-subscribed-notifications]. In
   addition, the following terms are introduced:

   o Datastore node: A node in the instantiated YANG data tree
      associated with a datastore. In this document, datastore nodes
      are often also simply referred to as "objects"

   o Datastore node update: A data item containing the current value of
      a datastore node at the time the datastore node update was
      created, as well as the path to the datastore node.

   o Datastore subscription: A subscription to a stream of datastore
      node updates.

   o Datastore subtree: A datastore node and all its descendant
      datastore nodes

   o On-change subscription: A datastore subscription with updates that
      are triggered when changes in subscribed datastore nodes are
      detected.

   o Periodic subscription: A datastore subscription with updates that
      are triggered periodically according to some time interval.

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

   o Update record: A representation of one or more datastore node
      updates. In addition, an update record may contain which type of
      update led to the datastore node update (e.g., whether the
      datastore node was added, changed, deleted). Also included in the
      update record may be other metadata, such as a subscription id of
      the subscription as part of which the update record was generated.
      In this document, update records are often also simply referred to
      as "updates".

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

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

3. Solution Overview

   This document specifies a solution that provides a subscription
   service for updates from a datastore. This solution supports dynamic
   as well as configured subscriptions to updates of datastore nodes.
   Subscriptions specify when notification messages (also referred to as
   "push updates") should be sent and what data to include in update
   records. Datastore node updates 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 YANG data model. 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 update triggers defining when to generate
   update records as well as what to include in an update record. Key
   enhancements include:

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

   o Specification of update policies contain conditions which trigger
      the generation and pushing of new update records. There are two
      types of subscriptions, distinguished by how updates are
      triggered: periodic and on-change.

      * For periodic subscriptions, the update 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", i.e. a reference point in time
         that can be used to calculate at which points in time periodic
         updates need to be assembled and sent.

      * For on-change subscriptions, an update trigger occurs whenever
         a change in the subscribed information is detected. Included
         are additional parameters that include:

         + 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
            resources in the publisher or receiver. In order to protect

against that, a dampening period MAY be used to specify the
interval which has to 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. This
means that when there is a change to one or more subscribed
objects, an update record containing those objects is
created immediately (when no dampening period is in effect)
or at the end of a dampening period (when a dampening period
is in fact in effect). If multiple changes to a single
object occur during a dampening period, only the value that
is in effect at the time when 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 an update when an object is created or
      deleted, but not when an object value changes).

   + Sync 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 of different parameters by a subscriber is to be
   discouraged. Therefore, in order to minimize the number of
   subscription iterations between subscriber and publisher, a dynamic
   subscription supports a simple negotiation between subscribers and
   publishers for subscription parameters. This negotiation is in the
   form of supplemental information which should 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 receivers 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, even if their values
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 publisher MAY suspend the subscription.

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 values that are current at the end of the
dampening period of all changed objects. 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, that
value (and not the interim 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 update record as part of an on-change subscription:

1. Just before a change, or at the start of a dampening period, evaluate any filtering and any access control rules to ensure receiver is authorized to view all subscribed datastore nodes (filtering out any nodes for which this is not the case). 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 an update record, which takes the form of YANG patch record [RFC8072] 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, even if the new value is no different from the original value (since changes that were made in the interim were canceled out). In case the changes involve creating a new datastore node, then deleting it, the YANG patch record will indicate deletion of the datastore node. Similarly, in case the changes involve deleting a new datastore node, then recreating it, the YANG patch record will indicate creation of the datastore node.

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, the subscriber has the option to create multiple subscriptions with different selection filters.

3.4. Reliability 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 datastore nodes 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. For this reason, the solution that is defined in this
document mandates that a publisher notifies receivers immediately and
reliably whenever it encounters a situation in which it is unable to
keep the terms of the subscription, and provides the publisher with
the option to suspend the subscription in such a case. This includes
indicating the fact that an update is incomplete as part of a push-
update or push-change-update notification, as well as emitting a
subscription-suspended notification as applicable. This is described
further in Section 3.11.1.

A publisher SHOULD reject a request for a subscription if it is
unlikely that the publisher will be able to 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
record corresponds to data that could have been read using a
retrieval operation.

3.5.2. On-Change Subscriptions

In an on-change subscription, update records 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 only configuration
objects.

A publisher indicates the type of change to a datastore node using
the different YANG patch operations: the "create" operation is used
for newly created objects (except entries in a user-ordered list),
the "delete" operation is used for deleted objects (including in
user-ordered lists), the "replace" operation is used when only the
object value changes, the "insert" operation is used when a new entry

is inserted in a list, and the "move" operation is used when an
existing entry in a user-ordered list is moved.

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 whether 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 indicates a "create" operation for a datastore node
where the receiver believes one already exists, or a "replace"
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 replaces any
   existing filter. The following selection filter types are included
   in the yang-push data model, and may be applied against a datastore:

   o 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 datastore 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. Consider an XPath filter which only
   passes a datastore node 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, then they are applied whenever a periodic
update record is constructed, and only datastore nodes that pass the
filter and 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" (see Section 3.7).

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 MUST 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 resync 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 over the
   course of a dampening period 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.)

   "Push-update" and "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
   interface (per [RFC8343]). 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">
    <id>1011</id>
    <datastore-contents>
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
         <interface>
           <name>eth0</name>
           <oper-status>up</oper-status>
         </interface>
        </interfaces>
    </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">
   <id>89</id>
   <datastore-changes>
     <yang-patch>
       <patch-id>0</patch-id>
       <edit>
         <edit-id>edit1</edit-id>
         <operation>replace</operation>
         <target>/ietf-interfaces:interfaces</target>
         <value>
           <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
             <interface>
               <name>eth0</name>
               <oper-status>down</oper-status>
             </interface>
           </interfaces>
         </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 '0'.
   Per [RFC8072], the 'patch-id' is an arbitrary string. With YANG
   Push, the publisher SHOULD put into the 'patch-id' a counter starting
   at '0' which increments with every 'push-change-update' generated for
   a subscription. If used as a counter, this counter MUST be reset to
   '0' 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 '0' after passing a maximum value of '4294967295' (i.e. maximum
   value that can be represented using uint32 data type). 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. Also, new error codes
   have been added that are able to indicate why a datastore
   subscription attempt has failed, along with new yang-data that MAY be
   used to include details on input parameters that might result in a
   successful subsequent RPC invocation.

The establishment or modification of 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.

In the case of a rejected request for an establishment of a datastore subscription, if there are hints, the hints SHOULD be transported within a yang-data "establish-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "establish-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "establish-subscription-datastore-error-info". All tree diagrams used in this document follow the notation defined in [RFC8340]

```
yang-data establish-subscription-datastore-error-info
   +--ro establish-subscription-datastore-error-info
      +--ro reason?                 identityref
      +--ro period-hint?            yang:timeticks
      +--ro filter-failure-hint?   string
      +--ro object-count-estimate? uint32
      +--ro object-count-limit?    uint32
      +--ro kilobytes-estimate?    uint32
      +--ro kilobytes-limit?       uint32
```

Figure 3: Tree diagram for establish-subscription-datastore-error-info

Similarly, in the case of a rejected request for modification of a datastore subscription, if there are hints, the hints SHOULD be transported within a yang-data "modify-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "modify-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "modify-subscription-datastore-error-info".

```
        yang-data modify-subscription-datastore-error-info
          +--ro modify-subscription-datasore-error-info
             +--ro reason?                  identityref
             +--ro period-hint?             yang:timeticks
             +--ro filter-failure-hint?     string
             +--ro object-count-estimate?   uint32
             +--ro object-count-limit?      uint32
             +--ro kilobytes-estimate?      uint32
             +--ro kilobytes-limit?         uint32
```

      Figure 4: Tree diagram for modify-subscription-datastore-error-info

3.9. Receiver Authorization

   A receiver of subscription data MUST only be sent updates for which
   it has 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.

   Each "push-update" and "push-change-update" MUST have access control
   applied, as is depicted in the following diagram. This includes
   validating that read access is permitted for any new objects selected
   since the last notification message was sent to a particular
   receiver. To accomplish this, implementations SHOULD support the
   conceptual authorization model of [RFC8341], specifically section
   3.2.4.

```
                     +-----------------+     +-------------------+
   push-update or -->│ datastore node  │ yes │ add datastore node│
   push-change-update│ access allowed? │ ---> │ to update record  │
                     +-----------------+     +-------------------+
```
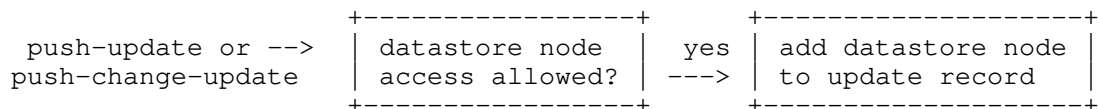
          Figure 5: Updated [RFC8341] access control for push updates

   A publisher MUST allow for the possibility that a subscription's
   selection filter references non-existent data or data that a receiver
   is not allowed to access. Such support permits a receiver the
   ability to monitor the entire lifecyle of some datastore tree without
   needing to explicitly enumerate every individual datastore node. If,
   after access control has been applied, there are no objects remaining
   in an update record, then (in case of a periodic subscription) only a
   single empty "push-update" notification MUST be sent. Empty "push-
   change-update" messages (in case of an on-change subscription) MUST
   NOT be sent. This is required to ensure that clients cannot

surreptitiously monitor objects that they do not have access to via
carefully crafted selection filters. By the same token, changes to
objects that are filtered MUST NOT affect any dampening intervals.

A publisher MAY choose to reject an establish-subscription request
which selects non-existent data or data that a receiver is not
allowed to access. As reason, the error identity "unchanging-
selection" SHOULD be returned. 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. In that case, the publisher
SHOULD include the error identity "unchanging-selection" as reason
when sending the "subscription-terminated" respectively
"subscription-suspended" notification. Such a capability enables the
publisher to avoid having to support continuous and total filtering
of a subscription's content for every update record. It also reduces
the possibility of leakage of access-controlled objects.

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 Datastore Nodes

In some cases, a publisher supporting on-change notifications may not
be able to push on-change updates for some object types. Reasons for
this might be that the value of the datastore node changes frequently
(e.g., [RFC8343]'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 SHOULD 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. In case the loss of an update is
unavoidable, it is critical that the receiver is notified
accordingly.

Update records for a single subscription MUST 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 act as follows.

o The publisher MUST set the "incomplete-update" flag on any update
   record which is known to be missing information.

o The publisher MAY choose to suspend the subscription as per
   [I-D.draft-ietf-netconf-subscribed-notifications]. If the
   publisher does not create an update record at all, it MUST suspend
   the subscription.

o 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 "sync-on-start" option is true 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 for the publisher to merge pending update records sent at
the same time.

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 update
   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. The tree diagram follows the notation defined
   in [RFC8340]. New schema objects defined here (i.e., beyond those
   from [I-D.draft-ietf-netconf-subscribed-notifications]) are
   identified with "yp". For the reader's convenience, in order to
   compact the tree representation, some nodes that are defined in ietf-
   subscribed-notifications and that are not essential to the
   understanding of the data model defined here have been removed. This
   is indicated by "..." in the diagram where applicable.

```
   module: ietf-subscribed-notifications
      ...
      +--rw filters
      |  ...
      |  +--rw yp:selection-filter* [filter-id]
      |     +--rw yp:filter-id                    string
      |     +--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* [id]
            |  ...
            +--rw (target)
            |  +--:(stream)
            |  |    ...
            |  +--:(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 (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:sync-on-start?     boolean
                   +--rw yp:excluded-change*   change-type

  rpcs:
    +---x establish-subscription
    |  +---w input
    |  |  ...
    |  |  +---w (target)
    |  |  |  +--:(stream)
    |  |  |  |  ...
    |  |  |  +--:(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 (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:sync-on-start?       boolean
   |  |              +---w yp:excluded-change*     change-type
   |  +--ro output
   |     +--ro id                             subscription-id
   |     +--ro replay-start-time-revision?   yang:date-and-time
   |             {replay}?
   +---x modify-subscription
   |  +---w input
   |      ...
   |      +---w (target)
   |      |  ...
   |      |  +--:(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 (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
   |  ...
   +---x kill-subscription
      ...

   yang-data (for placement into rpc error responses)
     ...

   notifications:
```

```
        +---n replay-completed {replay}?
        |  ...
        +---n subscription-completed
        |  ...
        +---n subscription-started {configured}?
        |  |  ...
        |  +--ro (target)
        |  |  ...
        |  |  +--:(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 (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:sync-on-start?      boolean
        |           +--ro yp:excluded-change*    change-type
        +---n subscription-resumed
        |  ...
        +---n subscription-modified
        |  ...
        |  +--ro (target)
        |  |  |  ...
        |  |  +--:(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 (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:sync-on-start?       boolean
         │            +--ro yp:excluded-change*     change-type
         +---n subscription-terminated
         │  ...
         +---n subscription-suspended
            ...

  module: ietf-yang-push

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

    yang-data: (for placement into rpc error responses)
      +-- resync-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 id?        sn:subscription-id
    │  +--ro datastore-contents?   <anydata>
    │  +--ro incomplete-update?      empty
    +---n push-change-update {on-change}?
        +--ro id?        sn:subscription-id
        +--ro datastore-changes?
        │   +--ro yang-patch
        │      +--ro patch-id         string
        │      +--ro ypatch:comment?     string
        │      +--ro ypatch:edit* [edit-id]
        │         +--ro ypatch:edit-id       string
        │         +--ro ypatch:operation     enumeration
        │         +--ro ypatch:target        target-resource-offset
        │         +--ro ypatch:point?        target-resource-offset
        │         +--ro ypatch:where?        enumeration
        │         +--ro ypatch:value?
        +--ro incomplete-update?     empty
```

                        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 parameters extending the basic subscription
   data model in [I-D.draft-ietf-netconf-subscribed-notifications]
   include:

   o The targeted datastore from which the selection is being made.
      The potential datastores include those from [RFC8341]. A platform
      may also choose to support a custom datastore.

   o 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.

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

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

   * an "anchor-time"; update intervals 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.

o 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" parameter 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 "sync-on-start" 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]. Notifications
   "subscription-started" and "subscription-modified" 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"
   notification.

An "id" (that identifies the subscription) MUST be transported along
with the subscribed contents. This allows a receiver to
differentiate which subscription resulted in a particular update
record.

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

An "incomplete-update" leaf. This leaf 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 provide 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 xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
        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 7: Establish-subscription RPC

   A positive response includes the "id" of the accepted subscription.
   In that case a publisher may respond:

```
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <id
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
        52
    </id>
</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 as part of the RPC error
   response depend on the specific transport that is used to manage the
   subscription. For example, in the case of NETCONF
   [I-D.draft-ietf-netconf-netconf-event-notifications], when a
   subscription request is rejected, the NETCONF RPC reply would be
   expected to include an "rpc-error" element with the following
   elements:

   o "error-type" of "application".

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

   o Optionally, an "error-severity" of "error".

   o Optionally, an "error-app-tag" with the value being a string that
      corresponds to an identity associated with the error, i.e. an
      identity with a base of "establish-subscription-error".

   o 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
       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"
   xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
   <rpc-error>
     <error-type>application</error-type>
     <error-tag>operation-failed</error-tag>
     <error-severity>error</error-severity>
     <error-path>/yp:periodic/yp:period</error-path>
        <error-info>
       <yp:establish-subscription-error-datastore>
         <yp:reason>yp:on-change-unsupported</yp:reason>
       </yp:establish-subscription-error-datastore>
        </error-info>
   </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 and datastore XPath filter
   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">
       <id>1011</id>
       <yp:datastore
       xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
         ds:operational
       </yp:datastore>
       <yp:datastore-xpath-filter
         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 to the subscription modification request.
If the request is rejected, the existing subscription is left
unchanged, and the publisher MUST send an RPC error response. This
response might have 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:

o "error-type" of "application".

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

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

o Optionally, an "error-app-tag" with the value being a string that
   corresponds to an identity associated with the error, i.e. an
   identity with a base of "modify-subscription-error".

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

o 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 "id".
This RPC is unmodified from
[I-D.draft-ietf-netconf-subscribed-notifications].

4.4.4. Resync-subscription RPC

   This RPC is supported 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">
   <resync-subscription
   xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push"
   xmlns:sn="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
     <id>1011</id>
   </resync-subscription>
 </netconf:rpc>
```

                          Resync 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 "resync-subscription-error".

4.4.5. YANG Module Synchronization

   To make subscription requests, the subscriber needs to know the YANG
   datastore schemas used by the publisher, which are available via the
   YANG Library module, ietf-yang-library.yang from [RFC7895]. The
   receiver is expected to know the YANG library information before
   starting a subscription.

   The set of modules, revisions, features, and deviations can change at
   run-time (if supported by the publisher implementation). For this
   purpose, the YANG library provides a simple "yang-library-change"
   notification that informs the subscriber that the library has
   changed. In this case, a subscription may need to be updated to take
   the updates into account. The receiver may also need to be informed
   of module changes in order to process updates regarding datastore
   nodes from changed modules correctly.

5. YANG Module

```
 <CODE BEGINS> file "ietf-yang-push@2018-10-22.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;
         reference
           "RFC 6991: Common YANG Data Types";
    }
    import ietf-subscribed-notifications {
      prefix sn;
      reference
        "draft-ietf-netconf-subscribed-notifications:
         Customized Subscriptions to a Publisher's Event Streams

         NOTE TO RFC Editor: Please replace above reference to
         draft-ietf-netconf-subscribed-notifications with RFC number
         when published (i.e. RFC xxxx).";
    }
    import ietf-datastores {
      prefix ds;
      reference
           "RFC 8342: Network Management Datastore Architecture (NMDA)";
    }
    import ietf-restconf   {
      prefix rc;
      reference
        "RFC 8040: RESTCONF Protocol";
    }

    import ietf-yang-patch {
      prefix ypatch;
      reference
        "RFC 8072: YANG Patch";
    }
    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.

     Copyright (c) 2018 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).

     This version of this YANG module is part of
     draft-ietf-netconf-yang-push-20; see the RFC itself for full
     legal notices.

     NOTE TO RFC EDITOR: Please replace above reference to
     draft-ietf-netconf-yang-push-20 with RFC number when published
     (i.e. RFC xxxx).";

  revision 2018-10-22 {
    description
      "Initial revision.
       NOTE TO RFC EDITOR:
       (1)Please replace the above revision date to
       the date of RFC publication when published.
       (2) Please replace the date in the file name
       (ietf-yang-push@2018-10-22.yang) to the date of RFC
       publication.
       (3) Please replace the following reference to
       draft-ietf-netconf-yang-push-20 with RFC number when
       published (i.e. RFC xxxx).";
    reference
      "draft-ietf-netconf-yang-push-20";
  }

/*
 * FEATURES
 */
```

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

 /*
  * IDENTITIES
  */

 /* Error type identities for datastore subscription  */

 identity resync-subscription-error {
    description
      "Problem found while attempting to fulfill an
      'resync-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-resync {
   base resync-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-sync-unsupported {
     base sn:establish-subscription-error;
     description
       "Neither sync on start nor resynchronization are supported
       for this subscription.  This error will be used for two
       reasons. First if an 'establish-subscription' RPC includes
       'sync-on-start', yet the publisher can't support sending a
       'push-update' for this subscription for reasons other than
       'on-change-unsupported' or 'sync-too-big'.  And second,
       if the 'resync-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 or dampening-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.";
   }

   identity update-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.";
   }

   identity sync-too-big {
     base sn:establish-subscription-error;
     base sn:modify-subscription-error;
     base resync-subscription-error;
     base sn:subscription-suspended-reason;
     description
       "Sync-on-start or resynchronization datatree exceeds a
       maximum size limit.  Hints on estimated size of what was too
       big may be returned as supplemental information.";
   }

   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
          "A change that refers to the creation of a new datastore
          node.";
      }
      enum "delete" {
        description
          "A change that refers to the deletion of a datastore
          node.";
      }
      enum "insert" {
        description
          "A change that refers to the insertion of a new
          user-ordered datastore node.";
      }
      enum "move" {
        description
          "A change that refers to a reordering of the target
          datastore node";
      }
      enum "replace" {
        description
          "A change that refers to a replacement of the target
          datastore node's value.";
      }
    }
    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:filter-id";
     }
     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.";
         reference
           "RFC 6241: Network Configuration Protocol, Section 6.";
```

```
        }
      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 'datastore-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 {
      when "../sn:target/yp:datastore";
          mandatory true;
      description
        "Defines necessary conditions for sending an event record to
        the subscriber.";
      case periodic {
        container periodic {
          presence "indicates a 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, in one hundredths of a second.";
          }
          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
                   6 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 sync-on-start {
          type boolean;
          default "true";
          description
            "When this object is set to false, it restricts an
            on-change subscription from sending push-update
            notifications.  When false, 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 true (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 resync 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 resync-subscription {
    if-feature "on-change";
    description
      "This RPC allows a subscriber of an active on-change
      subscription to request a full push of objects.
      A successful invocation results in a push-update of all
      datastore nodes that the subscriber is permitted to access.
      This RPC can only be invoked on the same session on which the
      subscription is currently active.  In case of an error, a
      resync-subscription-error is sent as part of an error
      response.";
    input {
      leaf id {
        type sn:subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be resynced.";
      }
    }
   }
```

```
   rc:yang-data resync-subscription-error {
     container resync-subscription-error {
       description
         "If a 'resync-subscription' RPC fails, the subscription is
         not resynced 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 resync-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" {
     when "sn:target/yp:datastore";
         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-datastore-error-info {
     container establish-subscription-datastore-error-info {
       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" {
    when "sn:target/yp:datastore";
    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-datastore-error-info {
    container modify-subscription-datastore-error-info {
      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 that would be subscribable as
      part of the NETCONF event stream by any receiver.";
    leaf id {
      type sn:subscription-id;
      description
        "This references the subscription which drove the
        notification to be sent.";
    }
    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 snapshot corresponds to the same
        snapshot that would be returned in a corresponding get
        operation with the same selection filter parameters
        applied.";
    }
    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.";
    }
  }
```

```
   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 id {
       type sn:subscription-id;
       description
         "This references the subscription which drove the
         notification to be sent.";
     }
     container datastore-changes {
       description
         "This contains the set of datastore changes of the
         target datastore starting at the time of the
         previous update, per the terms of the subscription.
         The datastore changes are encoded per RFC 8027
         (YANG Patch).";
           uses ypatch:yang-patch;
     }
     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.";
     }
   }

   augment "/sn:subscription-started" {
     description
       "This augmentation adds 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 the selection filter and where it
            originated from. 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 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 "filter-id";
        description
          "A list of pre-configured filters that can be applied
          to datastore subscriptions.";
        leaf filter-id {
          type string;
          description
            "An identifier to differentiate between selection
            filters.";
        }
        uses selection-filter-types;
      }
    }

    augment "/sn:subscriptions/sn:subscription" {
      when "sn:target/yp:datastore";
      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-20.txt (RFC form)

7. Security Considerations

The YANG module specified in this document defines a schema for data
that is designed to be accessed via network management protocols such
as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer
is the secure transport layer, and the mandatory-to-implement secure
transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer
is HTTPS, and the mandatory-to-implement secure transport is TLS
[RFC5246].

The Network Configuration Access Control Model (NACM) [RFC8341]
provides the means to restrict access for particular NETCONF or
RESTCONF users to a preconfigured subset of all available NETCONF or
RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that are
writable/creatable/deletable (i.e., config true, which is the
default). These data nodes may be considered sensitive or vulnerable
in some network environments. Write operations (e.g., edit-config)
to these data nodes without proper protection can have a negative
effect on network operations. These are the subtrees and data nodes
and their sensitivity/vulnerability. (It should be noted that the
YANG module augments the YANG module from
[I-D.draft-ietf-netconf-subscribed-notifications]. All security
considerations that are listed there are relevant also for datastore
subscriptions. In the following, we focus on the data nodes that are
newly introduced here.)

o Subtree "selection-filter" under container "filters": This subtree
   allows to specify which objects or subtrees to include in a
   datastore subscription. An attacker could attempt to modify the
   filter. For example, the filter might be modified to result in
   very few objects being filtered in order to attempt to overwhelm
   the receiver. Alternatively, the filter might be modified to
   result in certain objects to be excluded from updates, in order to
   have certain changes go unnoticed.

o Subtree "datastore" in choice "target" in list "subscription":
   Analogous to "selection filter", an attacker might attempt to
   modify the objects being filtered in order to overwhelm a receiver
   with a larger volume of object updates than expected, or to have
   certain changes go unnoticed.

o Choice "update-trigger" in list "subscription": By modifying the
   update trigger, an attacker might alter the updates that are being

         sent in order to confuse a receiver, to withhold certain updates
         to be sent to the receiver, and/or to overwhelm a receiver. For
         example, an attacker might modify the period with which updates
         are reported for a periodic subscription, or it might modify the
         dampening period for an on-change subscription, resulting in
         greater delay of successive updates (potentially affecting
         responsiveness of applications that depend on the updates) or in a
         high volume of updates (to exhaust receiver resources).

   o RPC "resync-subscription": This RPC allows a subscriber of an on-
      change subscription to request a full push of objects in the
      subscription's scope. This can result in a large volume of data.
      An attacker could attempt to use this RPC to exhaust resources on
      the server to generate the data, and attempt to overwhelm a
      receiver with the resulting data volume.

8. Acknowledgments

   For their valuable comments, discussions, and feedback, we wish to
   acknowledge Tim Jenkins, Martin Bjorklund, Kent Watsen, Susan Hares,
   Yang Geng, Peipei Guo, Michael Scharf, Guangying Zheng, Tom Petch,
   Henk Birkholz, Reshad Rahman, Qin Wu, Rohit Ranade, and Rob Wilton.

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-13
             (work in progress), August 2018.

   [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>.

   [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types",
              RFC 6991, DOI 10.17487/RFC6991, July 2013,
              <https://www.rfc-editor.org/info/rfc6991>.

   [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>.

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

   [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>.

   [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>.

   [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration
              Access Control Model", STD 91, RFC 8341,
              DOI 10.17487/RFC8341, March 2018,
              <https://www.rfc-editor.org/info/rfc8341>.

   [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
              and R. Wilton, "Network Management Datastore Architecture
              (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018,
              <https://www.rfc-editor.org/info/rfc8342>.

9.2. Informative References

   [I-D.draft-ietf-netconf-netconf-event-notifications]
              Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard,
              E., and A. Tripathy, "NETCONF Support for Event
              Notifications", August 2018.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246,
              DOI 10.17487/RFC5246, August 2008,
              <https://www.rfc-editor.org/info/rfc5246>.

   [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>.

   [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure
              Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011,
              <https://www.rfc-editor.org/info/rfc6242>.

   [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>.

   [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams",
              BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018,
              <https://www.rfc-editor.org/info/rfc8340>.

   [RFC8343] Bjorklund, M., "A YANG Data Model for Interface
              Management", RFC 8343, DOI 10.17487/RFC8343, March 2018,
              <https://www.rfc-editor.org/info/rfc8343>.

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        update-too-big
    on-change-unsupported            sync-too-big
    on-change-sync-unsupported       unchanging-selection
    period-unsupported
    update-too-big                   resync-subscription
    sync-too-big                     -------------------
    unchanging-selection             no-such-subscription-resync
                                     sync-too-big


   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 resync-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 "resync-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   update-too-big
    sn:suspension-timeout   synchronization-size
    unchanging-selection

Appendix B. Changes Between Revisions

   (To be removed by RFC editor prior to publication)

   v19 - v20

   o Minor updates per WGLC comments.

   v18 - v19

   o Minor updates per WGLC comments.

   v17 - v18

   o Minor updates per WGLC comments.

   v16 - v17

   o Minor updates to YANG module, incorporating comments from Tom
      Petch.

   o Updated references.

   v15 - v16

   o Updated security considerations.

   o Updated references.

   o Addressed comments from last call review, specifically comments
      received from Martin Bjorklund.

v14 - v15

o 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

o Minor text fixes.

v12 - v13

o Hint negotiation models now show error examples.

o yang-data structures for rpc errors.

v11 - v12

o Included Martin's review clarifications.

o QoS moved to subscribed-notifications

o time-of-update removed as it is redundant with RFC5277's
   eventTime, and other times from notification-messages.

o Error model moved to match existing implementations

o On-change notifiable removed, how to do this is implementation
   specific.

o NMDA model supported. Non NMDA version at https://github.com/
   netconf-wg/yang-push/

v10 - v11

o Promise model reference added.

o Error added for no-such-datastore

o Inherited changes from subscribed notifications (such as optional
   feature definitions).

o scrubbed the examples for proper encodings

v09 - v10

o Returned to the explicit filter subtyping of v00-v05

o identityref to ds:datastore made explicit

o Returned ability to modify a selection filter via RPC.

v08 - v09

o Minor tweaks cleaning up text, removing appendicies, and making
   reference to revised-datastores.

o Subscription-id (now:id) optional in push updates, except when
   encoded in RFC5277, Section 4 one-way notification.

o Finished adding the text desciding the resync subscription RPC.

o Removed relationships to other drafts and future technology
   appendicies as this work is being explored elsewhere.

o Deferred the multi-line card issue to new drafts

o Simplified the NACM interactions.

v07 - v08

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

v06 - v07

o Clarifying text tweaks.

o Clarification that filters act as selectors for subscribed
   datastore nodes; support for value filters not included but
   possible as a future extension

o Filters don't have to be matched to existing YANG objects

v05 - v06

o Security considerations updated.

o Base YANG model in [subscribe] updated as part of move to
   identities, YANG augmentations in this doc matched up

o Terms refined and text updates throughout

o Appendix talking about relationship to other drafts added.

o Datastore replaces stream

o Definitions of filters improved

v04 to v05

o Referenced based subscription document changed to Subscribed
   Notifications from 5277bis.

o Getting operational data from filters

o Extension notifiable-on-change added

o New appendix on potential futures. Moved text into there from
   several drafts.

o Subscription configuration section now just includes changed
   parameters from Subscribed Notifications

o Subscription monitoring moved into Subscribed Notifications

o New error and hint mechanisms included in text and in the yang
   model.

o Updated examples based on the error definitions

o Groupings updated for consistency

o Text updates throughout

v03 to v04

o Updates-not-sent flag added

o Not notifiable extension added

o Dampening period is for whole subscription, not single objects

o Moved start/stop into rfc5277bis

o Client and Server changed to subscriber, publisher, and receiver

o Anchor time for periodic

o Message format for synchronization (i.e. sync-on-start)

o Material moved into 5277bis

o QoS parameters supported, by not allowed to be modified by RPC

   o   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