```
NETCONF                                             A. Gonzalez Prieto
Internet-Draft                                                A. Clemm
Intended status: Standards Track                               E. Voit
Expires: September 22, 2016                         E. Nilsen-Nygaard
                                                            A. Tripathy
                                                          Cisco Systems
                                                         March 21, 2016
```

NETCONF Event Notifications
draft-gonzalez-netconf-5277bis-01

Abstract

This document defines capabilities and operations for providing asynchronous message notification delivery on top of the Network Configuration protocol (NETCONF). This notification delivery is an optional capability built on top of the base NETCONF definition. This document is eventually intended to obsolete RFC 5277.

Status of This Memo

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Table of Contents

1. Introduction

[RFC6241] can be conceptually partitioned into four layers:

```
         Layer                         Example
       +-------------+     +-------------------------------------------+
       |   Content   |     |     Configuration data                    |
       +-------------+     +-------------------------------------------+
             |                           |
       +-------------+     +-------------------------------------------+
       | Operations  |     |<get-config>, <edit-config>, <notification>|
       +-------------+     +-------------------------------------------+
             |                           |                       |
       +-------------+     +-----------------------------+       |
       |     RPC     |     |    <rpc>, <rpc-reply>       |       |
       +-------------+     +-----------------------------+       |
             |                           |                       |
       +-------------+     +-------------------------------------------+
       |  Transport  |     |   BEEP, SSH, SSL, console                 |
       |  Protocol   |     |                                           |
       +-------------+     +-------------------------------------------+
```

This document defines mechanisms that provide an asynchronous message notification delivery service for the NETCONF protocol [RFC6241]. This is an optional capability built on top of the base NETCONF definition. This document also defines the capabilities and operations necessary to establish, monitor, and support subscriptions into this notification delivery service.

1.1. Motivation

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

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

- o Each subscription requires a separate NETCONF connection, which is wasteful.

- o The only mechanism to terminate a subscription is terminating the underlying NETCONF connection.

- o No ability to modify subscriptions once they have been created.

- o No ability to notify the receiver of a subscription if the server is dropping events.

- o No mechanism to monitor subscriptions.

- No alternative mechanism to create subscriptions via RPCs. Thus the lifetime of the subscription is limited by that of the underlaying NETCONF session.

- Predates YANG and defines RPCs, notifications, and data nodes outside of the YANG framework.

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

- Ability to dynamically or statically subscribe to event notifications available on a NETCONF agent.

- Ability to negotiate acceptable dynamic subscription parameters.

- Ability to support multiple subscriptions over a single NETCONF session.

- Ability to filter the subset of notifications to be pushed with stream-specific semantics.

- Ability for the notification payload to be interpreted independently of the NETCONF transport protocol. (In other words, the encoded notification fully describes itself.)

- Mechanism to communicate the notifications.

- Ability to replay locally logged notifications.

- Backwards compatible with RFC 5277 implementations.

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

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Event: Something that happens that may be of interest. (e.g., a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.)

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

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

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

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

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

Filter: Evaluation criteria, which may be applied against a targeted set of objects/events in a subscription. Information traverses the filter only if specified filter criteria are met.

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

Static subscription: A subscription installed via a configuration interface.

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

NACM: NETCONF Access Control Model.

RPC: Remote Procedure Call.

1.3. Solution Overview

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

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

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

The solution supports subscribing to event notifications using two mechanisms.

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

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

Some key characteristics of static and dynamic subscriptions include:

- The lifetime of a dynamic subscription is limited by the lifetime of the subscriber session used to establish it. Typically loss of the transport session tears down any dependent dynamic subscriptions.

- The lifetime of a static subscription is driven by configuration being present on the running configuration. This implies static subscriptions persist across reboots, and persists even when transport is unavailable. This also means static subscriptions do not support negotiation.

- Subscriptions can be modified or terminated at any point of their lifetime. Static subscriptions can be modified by any configuration client with write rights on the configuration of the subscription.

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

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

The NETCONF agent may decide to terminate a dynamic subscription at any time. Similarly the NETCONF agent may decide to temporarily suspend the sending of event notifications for either static or dynamic subscriptions. Such termination or suspension may be driven by the agent running out of resources to serve the subscription, or by internal errors on the server.

2. Solution

2.1. Event Streams

An event stream is a set of events available for subscription from a NETCONF server. It is out of the scope of this document to identify a) how streams are defined, b) how events are defined/generated, and c) how events are assigned to streams.

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

2.2. Event Stream Discovery

A NETCONF client can retrieve the list of available event streams from a NETCONF server using the <get> operation. The reply contains the elements defined in the YANG model under the container /netconf/streams, which includes the name and description of the streams.

The following example shows retrieving the list of available event stream list using the <get> operation.

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter type="subtree">
            <netconf
                 xmlns="urn:ietf:params:xml:ns:netmod:notification">
                <streams/>
            </netconf>
        </filter>
     </get>
</rpc>
```

Figure 1: Get streams

The NETCONF server returns a list of event streams available for subscription. In this example, the list contains the NETCONF, SNMP, and syslog-critical streams.

```
<rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
        <netconf
               xmlns="urn:ietf:params:xml:ns:netmod:notification">
            <streams>
                <stream>
                    <name>
                        NETCONF
                    </name>
                    <description>
                        default NETCONF event stream
                    </description>
                    <replaySupport>
                        true
                    </replaySupport>
                    <replayLogCreationTime>
                        2016-02-05T00:00:00Z
                    </replayLogCreationTime>
                </stream>
                <stream>
                    <name>
                        SNMP
                    </name>
                    <description>
                        SNMP notifications
                    </description>
                    <replaySupport>
                        false
                    </replaySupport>
                </stream>
                <stream>
                    <name>
                        syslog-critical
                    </name>
                    <description>
                        Critical and higher severity
                    </description>
                    <replaySupport>
                        true
                    </replaySupport>
                    <replayLogCreationTime>
                        2007-07-01T00:00:00Z
                    </replayLogCreationTime>
```

```
                   </stream>
              </streams>
           </netconf>
       </data>
   </rpc-reply>
```

Figure 2: Get streams response

2.3. Default Event Stream

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

2.4. Data Model Trees for Event Notifications

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

2.4.1. Data Model Tree for RFC5277 (netconf namespace)

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

2.4.2. Data Model Tree for RFC5277 (netmod namespace)

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

2.4.3. Data Model for RFC5277-bis Extensions

```
   module: ietf-event-notifications
      +--rw filters
      |  +--rw filter* [filter-id]
      |     +--rw filter-id    filter-id
      |     +--rw (filter-type)?
      |        +--:(rfc5277)
      |           +--rw filter
      +--rw subscription-config {configured-subscriptions}?
      |  +--rw subscription* [subscription-id]
      |     +--rw subscription-id     subscription-id
      |     +--rw stream?             string
      |     +--rw (filter-type)?
      |     |  +--:(rfc5277)
      |     |  |  +--rw filter
      |     |  +--:(by-reference)
      |     |     +--rw filter-ref?         filter-ref
      |     +--rw startTime?          yang:date-and-time
      |     +--rw stopTime?           yang:date-and-time
      |     +--rw encoding?           encoding
      |     +--rw receivers
      |     |  +--rw receiver* [address]
      |     |     +--rw address     inet:host
      |     |     +--rw port        inet:port-number
      |     |     +--rw protocol?   transport-protocol
      |     +--rw (push-source)?
      |        +--:(interface-originated)
      |        |  +--rw source-interface?   if:interface-ref
      |        +--:(address-originated)
      |           +--rw source-vrf?         uint32
      |           +--rw source-address      inet:ip-address-no-zone
      +--ro subscriptions
         +--ro subscriptions* [subscription-id]
            +--ro subscription-id            subscription-id
            +--ro (subscription-type)?
            |  +--:(via-configuration)
            |  |  +--ro configured-subscription?
                                empty {configured-subscriptions}?
            |  +--:(created-via-RPC)
            |     +--ro created-subscription?      empty
            +--ro subscription-status?       identityref
            +--ro stream?                    string
            +--ro (filter-type)?
            |  +--:(rfc5277)
            |  |  +--ro filter
            |  +--:(by-reference)
            |     +--ro filter-ref?                filter-ref
            +--ro startTime?                 yang:date-and-time
            +--ro stopTime?                  yang:date-and-time
            +--ro encoding?                  encoding
            +--ro receivers
            |  +--ro receiver* [address]
            |     +--ro address     inet:host
            |     +--ro port        inet:port-number
            |     +--ro protocol?   transport-protocol
            +--ro (push-source)?
               +--:(interface-originated)
               |  +--ro source-interface?       if:interface-ref
               +--:(address-originated)
                  +--ro source-vrf?             uint32
                  +--ro source-address          inet:ip-address-no-zone
   augment /netmod-notif:replayComplete:
      +--ro subscription-id?   subscription-id
   augment /netmod-notif:notificationComplete:
      +--ro subscription-id?   subscription-id
   augment /netmod-notif:netconf/netmod-notif:streams:
      +--rw exclude-from-NETCONF-stream?   empty
   rpcs:
      +---x establish-subscription
      |  +--ro input
      |  |  +--ro stream?       string
      |  |  +--ro (filter-type)?
      |  |  |  +--:(rfc5277)
      |  |  |  |  +--ro filter
      |  |  |  +--:(by-reference)
      |  |  |     +--ro filter-ref?   filter-ref
      |  |  +--ro startTime?    yang:date-and-time
      |  |  +--ro stopTime?     yang:date-and-time
      |  |  +--ro encoding?     encoding
      |  +--ro output
      |     +--ro subscription-result    subscription-result
      |     +--ro (result)?
      |        +--:(success)
      |        |  +--ro subscription-id        subscription-id
      |        +--:(no-success)
      |           +--ro stream?                string
      |           +--ro (filter-type)?
      |           |  +--:(rfc5277)
      |           |  |  +--ro filter
      |           |  +--:(by-reference)
      |           |     +--ro filter-ref?            filter-ref
      |           +--ro startTime?             yang:date-and-time
      |           +--ro stopTime?              yang:date-and-time
      |           +--ro encoding?              encoding
      +---x modify-subscription
      |  +--ro input
      |  |  +--ro subscription-id?   subscription-id
      |  |  +--ro stream?            string
      |  |  +--ro (filter-type)?
      |  |  |  +--:(rfc5277)
      |  |  |  |  +--ro filter
      |  |  |  +--:(by-reference)
      |  |  |     +--ro filter-ref?        filter-ref
      |  |  +--ro startTime?         yang:date-and-time
      |  |  +--ro stopTime?          yang:date-and-time
      |  |  +--ro encoding?          encoding
      |  +--ro output
      |     +--ro subscription-result    subscription-result
      |     +--ro stream?                string
      |     +--ro (filter-type)?
      |     |  +--:(rfc5277)
      |     |  |  +--ro filter
      |     |  +--:(by-reference)
      |     |     +--ro filter-ref?            filter-ref
      |     +--ro startTime?             yang:date-and-time
      |     +--ro stopTime?              yang:date-and-time
      |     +--ro encoding?              encoding
      +---x delete-subscription
         +--ro input
            +--ro subscription-id?   subscription-id
   notifications:
      +---n subscription-started
      |  +--ro subscription-id    subscription-id
      |  +--ro stream?            string
      |  +--ro (filter-type)?
      |  |  +--:(rfc5277)
      |  |  |  +--ro filter
      |  |  +--:(by-reference)
      |  |     +--ro filter-ref?        filter-ref
      |  +--ro startTime?         yang:date-and-time
      |  +--ro stopTime?          yang:date-and-time
      |  +--ro encoding?          encoding
      +---n subscription-suspended
      |  +--ro subscription-id    subscription-id
      |  +--ro reason?            subscription-susp-reason
      +---n subscription-resumed
      |  +--ro subscription-id    subscription-id
      +---n subscription-modified
      |  +--ro subscription-id    subscription-id
      |  +--ro stream?            string
      |  +--ro (filter-type)?
      |  |  +--:(rfc5277)
      |  |  |  +--ro filter
      |  |  +--:(by-reference)
      |  |     +--ro filter-ref?        filter-ref
      |  +--ro startTime?         yang:date-and-time
      |  +--ro stopTime?          yang:date-and-time
      |  +--ro encoding?          encoding
      +---n subscription-terminated
      |  +--ro subscription-id    subscription-id
      |  +--ro reason?            subscription-term-reason
      +---n added-to-subscription
      |  +--ro subscription-id    subscription-id
      |  +--ro stream?            string
      |  +--ro (filter-type)?
      |  |  +--:(rfc5277)
      |  |  |  +--ro filter
      |  |  +--:(by-reference)
      |  |     +--ro filter-ref?        filter-ref
      |  +--ro startTime?         yang:date-and-time
      |  +--ro stopTime?          yang:date-and-time
      |  +--ro encoding?          encoding
      +---n removed-from-subscription
         +--ro subscription-id    subscription-id
```

2.5. Creating a Subscription

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

This operation is included in the document for supporting backwards compatibility with [RFC5277] clients. New clients are expected not to use this operation, but establish subscriptions as defined in Section 2.6

2.5.1. Parameters

The input parameters of the operation are:

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

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

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

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

2.5.2. Usage Example

The following demonstrates dynamically creating a simple subscription.

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

Figure 3: Create subscription

2.5.3. Positive Response

If the NETCONF server can satisfy the request, the server sends an <ok> element.

```
<netconf:rpc netconf:message-id="102"
       xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
    <create-subscription
          xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
        <filter netconf:type="xpath"
                xmlns:ex="http://example.com/event/1.0"
             select="/ex:event[ex:eventClass='fault' and
                      (ex:severity='minor' or ex:severity='major'
                       or ex:severity='critical')]"/>
    </create-subscription>
</netconf:rpc>

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

Figure 4: Successful create subscription

2.5.4. Negative Response

If the request cannot be completed for any reason, an <rpc-error> element is included within the <rpc-reply>. Subscription requests can fail for several reasons including if a filter with invalid syntax is provided or if the name of a non-existent stream is provided.

If a stopTime is specified in a request without having specified a startTime, the following error is returned:

Tag: missing-element
Error-type: protocol
Severity: error
Error-info: <bad-element>: startTime
Description: An expected element is missing.

If the optional replay feature is requested but the NETCONF server does not support it, the following error is returned:

Tag: operation-failed
Error-type: protocol
Severity: error
Error-info: none
Description: Request could not be completed because the
             requested operation failed for some reason
             not covered by any other error condition.

If a stopTime is requested that is earlier than the specified startTime, the following error is returned:

```
Tag: bad-element
Error-type: protocol
Severity: error
Error-info: <bad-element>: stopTime
Description: An element value is not correct;
             e.g., wrong type, out of range, pattern mismatch.
```

If a startTime is requested that is later than the current time, the following error is returned:

```
Tag: bad-element
Error-type: protocol
Severity: error
Error-info: <bad-element>: startTime
Description: An element value is not correct;
             e.g., wrong type, out of range, pattern mismatch.
```

2.6. Establishing a Subscription

This operation is an evolution of the create subscription operation. It allows a subscriber to request the creation of a subscription both via RPC and configuration operations. When invoking the RPC, establish-subscription permits negotiating the subscription terms, changing them dynamically and enabling multiple subscriptions overs a single NETCONF session (if interleaving [RFC6241] is supported), and canceling subscriptions without terminating the NETCONF session.

2.6.1. Parameters

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

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

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

2.6.2. Usage Example

The following demonstrates establishing a simple subscription.

```
   <netconf:rpc message-id="101"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
       <establish-subscription
               xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
       </establish-subscription>
   </netconf:rpc>
```

Figure 5: Establish subscription

2.6.3. Positive Response

If the NETCONF server can satisfy the request, the server sends a positive <subscription-result> element, and the subscription-id of the accepted subscription.

```
   <netconf:rpc netconf:message-id="102"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
       <establish-subscription
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           <filter netconf:type="xpath"
                   xmlns:ex="http://example.com/event/1.0"
                select="/ex:event[ex:eventClass='fault' and
                         (ex:severity='minor' or ex:severity='major'
                          or ex:severity='critical')]"/>
       </establish-subscription>
   </netconf:rpc>

   <rpc-reply message-id="102"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <subscription-result
             xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
          ok
       </subscription-result>
       <subscription-id
             xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
          52
       </subscription-id>
   </rpc-reply>
```

Figure 6: Successful establish subscription

2.6.4. Negative Response

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

If the client has no authorization to establish the subscription, the <subscription-result> indicates an authorization error. For instance:

```
<netconf:rpc netconf:message-id="103"
       xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
    <establish-subscription
           xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
        <stream>foo</stream>
    </establish-subscription>
</netconf:rpc>

<rpc-reply message-id="103"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <subscription-result
           xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
        error-data-not-authorized
    </subscription-result>
</rpc-reply>
```

Figure 7: Unsuccessful establish subscription

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

```
<netconf:rpc message-id="101"
         xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
      <establish-subscription
            xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
         <stream>push-update</stream>
         <filter netconf:type="xpath"
                  xmlns:ex="http://example.com/sample-data/1.0"
               select="/ex:foo"/>
         <period
             xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
            500
         </period>
         <encoding>encode-xml</encoding>
      </establish-subscription>
</netconf:rpc>


   <rpc-reply message-id="101"
         xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <subscription-result
           xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
        error-insufficient-resources
      </subscription-result>
      <period
           xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
        2000
      </period>
   </rpc-reply>
```

Figure 8: Subscription establishment negotiation

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

2.7. Modifying a Subscription

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

Dynamic subscriptions established via RPC can only be modified (or deleted) via RPC using the same session used to establish it. Configuration-based (i.e., static) subscriptions cannot be modified (or deleted) using RPCs. Instead, configured subscriptions are modified (or deleted) as part of regular configuration operations. Servers MUST reject any attempts to modify (or delete) static subscriptions via RPC.

2.7.1. Parameters

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

2.7.2. Usage Example

The following demonstrates modifying a subscription. Consider a subscription from [netconf-yang-push], which augments the establish-subscription with some additional parameters, including "period". A subscription may be established as follows.

```
   <netconf:rpc message-id="101"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
       <establish-subscription
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           <stream>push-update</stream>
           <filter netconf:type="xpath"
               xmlns:ex="http://example.com/sample-data/1.0"
               select="/ex:foo"/>
           <period
                 xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
               500
           </period>
           <encoding>encode-xml</encoding>
       </establish-subscription>
   </netconf:rpc>

   <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <subscription-result
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           ok
       </subscription-result>
       <subscription-id
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           1922
       </subscription-id>
   </rpc-reply>
```

Figure 9: Establish subscription to be modified

The subscription may be modified with:

```
   <netconf:rpc message-id="102"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
       <modify-subscription
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           <subscription-id>1922</subscription-id>
           <period>1000</period>
       </modify-subscription >
   </netconf:rpc>

   <rpc-reply message-id="102"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <subscription-result
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           ok
       </subscription-result>
       <subscription-id
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           1922
       </subscription-id>
   </rpc-reply>
```

Figure 10: Modify subscription

2.7.3. Positive Response

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

```
   <netconf:rpc message-id="102"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
       <modify-subscription
            xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           <subscription-id>1922</subscription-id>
           <period
               xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
             1000
           </period>
       </modify-subscription >
   </netconf:rpc>

   <rpc-reply message-id="102"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <subscription-result
            xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           ok
       </subscription-result>
   </rpc-reply>
```

Figure 11: Successful modify subscription

2.7.4. Negative Response

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

```
   <netconf:rpc message-id="102"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
       <modify-subscription
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           <subscription-id>1922</subscription-id>
           <period
                 xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
               100
           </period>
       </modify-subscription>
   </netconf:rpc>

   <rpc-reply message-id="102"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <subscription-result
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           error-insufficient-resources
       </subscription-result>
       <period>500</period>
   </rpc-reply>
```

Figure 12: Unsuccessful modify subscription

2.8. Deleting a Subscription

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

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

2.8.1. Parameters

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

2.8.2. Usage Example

The following demonstrates deleting a subscription.

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

Figure 13: Delete subscription

2.8.3. Positive Response

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

```
<netconf:rpc message-id="103"
       xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
    <delete-subscription
           xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
        <subscription-id>1922</subscription-id>
    </delete-subscription>
</netconf:rpc>

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

Figure 14: Successful delete subscription

2.8.4. Negative Response

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

```
   <netconf:rpc message-id="103"
          xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
       <delete-subscription
             xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
           <subscription-id>2017</subscription-id>
       </delete-subscription>
   </netconf:rpc>

   <rpc-reply message-id="101"
          xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <rpc-error>
           <error-type>application</error-type>
           <error-tag>invalid-value</error-tag>
           <error-severity>error</error-severity>
           <error-path
              xmlns:t="urn:ietf:params:xml:ns:netconf:notification:1.1">
             /t:subscription-id
           </error-path>
           <error-message xml:lang="en">
               Subscription-id 2017 does not exist
           </error-message>
       </rpc-error>
   </rpc-reply>
```

Figure 15: Unsuccessful delete subscription

2.9. Static Subscriptions

A static subscription is a subscription installed via a configuration interface.

Static subscriptions persist across reboots, and persist even when transport is unavailable. This also means static subscriptions do not support negotiation.

Static subscriptions can be modified by any configuration client with write rights on the configuration of the subscription. Subscriptions can be modified or terminated at any point of their lifetime.

Supporting static subscriptions is optional and advertised using the "configured-subscriptions" feature.

2.9.1. Creating a Static Subscription

Static subscriptions cannot be created via configuration operations. New clients should use the mechanisms described in Section 2.9.2 for establishing static subscriptions.

2.9.2. Establishing a Static Subscription

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

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

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

```
<rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
       xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <subscription-config
            xmlns="urn:ietf:params:xml:ns:netconf:notification:1.1">
            <subscription>
                <subscription-id>
                    1922
                </subscription-id>
                <stream>
                    foo
                </stream>
                <receiver>
                    <address>
                        1.2.3.4
                    </address>
                    <port>
                        1234
                    </port>
                </receiver>
            </subscription>
        </subscription-config>
    </edit-config>
</rpc>
```

Figure 16: Establish static subscription

if the request is accepted, the server would reply:

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

Figure 17: Response to a successful static subscription establishment

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

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

Figure 18: Response to a failed static subscription establishment

2.9.3. Modifying a Static Subscription

Static subscriptions can be modified using configuration operations against the top-level subtree subscription-config.

Immediately after a subscription is successfully modified, the server sends to the existing receivers a control-plane notification stating the subscription has been modified (i.e., subscription-modified).

If the modification involved adding and/or removing receivers, those modified receivers are sent control-plane notifications, indicating they have been added (i.e, added-to-subscription, with the same contents as a modified-subscription) or removed (i.e., removed-from-subscription)

2.9.4. Deleting a Static Subscription

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

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

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

Figure 19: Deleting a static subscription

Immediately after a subscription is successfully deleted, the server sends to the receivers a control-plane notification stating the subscription has been terminated (subscription-terminated).

2.10. Event (Data Plane) Notifications

Once a subscription has been set up, the NETCONF server sends (asynchronously) the event notifications from the subscribed stream. We refer to these as data plane notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the NETCONF session used to create or establish the subscription. For static subscriptions, event notifications are sent over the specified connections.

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

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

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

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

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

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

Figure 20: Definition of a data plane notification

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

Figure 21: Data plane notification

The equivalent using json encoding would be

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

Figure 22: Data plane notification using JSON encoding

2.11. Control Plane Notifications

In addition to data plane notifications, a server may send control plane notifications to indicate to receivers that an event related to the subscription management has occurred. Control plane notifications cannot be filtered out. Types of control plane notifications include:

2.11.1. replayComplete

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

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

2.11.2. notificationComplete

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

2.11.3. subscription-started

This notification indicates that a static subscription has started and data updates are beginning to be sent. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

2.11.4. subscription-modified

This notification indicates that a static subscription has been modified successfully. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

2.11.5. subscription-terminated

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

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

2.11.6. subscription-suspended

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

2.11.7. subscription-resumed

This notification indicates that a previously suspended dubscription has been resumed. Data plane notifications generated in the future will be sent after the subscription terms. Resumptions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of static subscriptions).

2.12. Subscription Management

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

```
                    .-------.
                    | start |
                    '-------'
                        |
               create   |  establish
                        |
                        |   .----------modify-------------.
                        v   v                              '
                  .-----------.                      .-----------.
       .--------. |           |------>suspend------->|           |
     modify      '|  active   |                      | suspended |
       '--------->|           |<----reactivate<------|           |
                  '-----------'                      '-----------'
                        |                                  |
                     delete                             delete
                        |                                  |
                        v                                  |
                    .-------.                              |
                    |  end  |<-----------------------------'
                    '-------'
```

Of interest in this state machine are the following:

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

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

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

3. Data Models for Event Notifications

3.1. Data Model for RFC5277 (netmod namespace)

```
<CODE BEGINS>
file "ietf-5277-netmod@2016-03-21.yang"
module ietf-5277-netmod {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-5277-netmod";
    // TODO: examples in the draft consider the namespace below
    //       which follows the namespace format in 5277
    // "urn:ietf:params:xml:ns:netmod:notification";
  prefix netmod-notif;
```

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

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


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

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

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

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

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

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

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

     revision 2016-03-21 {
       description
         "Model for data nodes and notifications in RFC 5277:
          NETCONF Event Notifications";
       reference
         "RFC 5277: NETCONF Event Notifications";
     }


     /*
      * EXTENSIONS
      */

     extension control-plane-notif {
       description
```

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


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

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

         list stream {
           must "1 = count(./name == 'NETCONF')" {
             description
               "The list must contain a NETCONF stream.
                A NETCONF server implementation supporting the
                notification capability MUST support the
                'NETCONF' notification event stream. This stream
                contains all NETCONF XML event notifications supported
                by the NETCONF server.
                The exact string 'NETCONF' is used during the
                advertisement of stream support during the <get>
                operation on <streams> and during the
                <create-subscription> operation.";
           }
           key "name";
           description
             "The list available event streams.";
           leaf name {
             type string;
             description
               "The name of the event stream.
                If this is the default NETCONF stream, this must have
                the value 'NETCONF'.";
           }
           leaf description {
             type string;
             mandatory true;
             description
               "A description of the event stream, including such
                information as the type of events that are sent over
                this stream.";
           }
           leaf replaySupport {
             type boolean;
             mandatory true;
             description
               "An indication of whether or not event replay is
                available on this stream.";
           }
           leaf replayLogCreationTime {
             when "../replaySupport" {
               description
                 "This object MUST be present if replay is supported.";
             }
             type yang:date-and-time;
             mandatory true;
             description
               "The timestamp of the creation of the log used to
                support the replay function on this stream.
                Note that this might be earlier then the earliest
                available notification in the log.  This object
                is updated if the log resets for some reason.
                This object MUST be present if replay is supported.";
           }
           leaf replayLogAgedTime {
             when "current()/../replaySupport" {
               description
                 "This object MUST be present if replay is supported
                 and any notifications have been aged out of the log.";
             }
             type yang:date-and-time;
             mandatory true;
             description
               "The timestamp of the last notification aged
                out of the log. This object MUST be present
                if replay is supported and any notifications
                have been aged out of the log.";
           }
         } // list stream
       } // container streams
     } // container netconf
     /*
      * NOTIFICATIONS
      */

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

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

   <CODE ENDS>
```

3.2. Data Model for RFC5277 (netconf namespace)

```
   <CODE BEGINS>
   file "ietf-5277-netconf@2016-03-21.yang"
   module ietf-5277-netconf {
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-5277-netconf";
       // TODO: examples in the draft consider the namespace below
       //       which follows the namespace format in 5277
       // "urn:ietf:params:xml:ns:netconf:notification:1.0";
     prefix notif;


     import ietf-yang-types {
       prefix yang;
     }

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

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

        WG Chair: Mehmet Ersue
```

```
                  <mailto:mehmet.ersue@nokia.com>

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

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

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

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

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

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

     revision 2016-03-21 {
       description
         "Model for RPC in RFC 5277: NETCONF Event Notifications";
       reference
         "RFC 5277: NETCONF Event Notifications";
     }


     /*
      * GROUPINGS
      */

     grouping base-filter {
       description
         "This grouping defines the base for filters for
          notification events.
          It includes the filter defined in 5277 and
          it enables extending filtering to other
          types of filters";
       choice filter-type {
         description
           "A filter needs to be a single filter of a given type.
            Mixing and matching of multiple filters does not occur
            at the level of this grouping.";
         case rfc5277 {
           anyxml filter {
             description
```

```
               "Filter per RFC 5277. Notification filter.
               If a filter element is specified to look for data of a
               particular value, and the data item is not present
               within a particular event notification for its value to
               be checked against, the notification will be filtered
               out. For example, if one were to check for
               'severity=critical' in a configuration event
               notification where this field was not supported, then
               the notification would be filtered out. For subtree
               filtering, a non-empty node set means that the filter
               matches.  For XPath filtering, the mechanisms defined
               in [XPATH] should be used to convert the returned
               value to boolean.";
           }
         }
       }
     }

     grouping subscription-info-5277 {
       description
         "This grouping describes the information in a 5277
          subscription.";
       leaf stream {
         type string;
         default "NETCONF";
         description
           "Indicates which stream of events is of interest.
            If not present, events in the default NETCONF stream
            will be sent.";
       }
       uses base-filter;
       leaf startTime {
         type yang:date-and-time;
         description
           "Used to trigger the replay feature
            and indicate that the replay should start at the time
            specified.  If <startTime> is not present, this is
            not a replay subscription.
            It is not valid to specify start
            times that are later than the current time.  If the
            <startTime> specified is earlier than the log can
            support, the replay will begin with the earliest available
            notification.  This parameter is of type dateTime and
            compliant to [RFC3339].  Implementations must
            support time zones.";
       }
       leaf stopTime {
         type yang:date-and-time;
         must "current()  > ../startTime" {
           description
             "stopTime must be used with and be later than
              <startTime>";
         }
         description
           "Used with the optional replay feature to indicate the
            newest notifications of interest. If <stopTime> is
            not present, the notifications will continue until the
            subscription is terminated.  Must be used with and be
            later than <startTime>.  Values of <stopTime>
            in the future are valid. This parameter is of type dateTime
            and compliant to [RFC3339].  Implementations must support
            time zones.";
       }
     }


     /*
      * RPCs
      */

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

   <CODE ENDS>
```

3.3. Data Model for RFC5277-bis Extensions

```
   <CODE BEGINS>
   file "ietf-event-notifications@2016-03-21.yang"
   module ietf-event-notifications {
     namespace
       "urn:ietf:params:xml:ns:yang:ietf-event-notifications";
       // TODO: examples in the draft consider the namespace below
       //       which follows the namespace format in 5277
       // "urn:ietf:params:xml:ns:netconf:notification:1.1";
     prefix notif-bis;
     import ietf-inet-types {
       prefix inet;
     }
     import ietf-5277-netmod {
       prefix netmod-notif;
     }
     import ietf-5277-netconf {
       prefix notif;
     }
     import ietf-interfaces {
       prefix if;
     }

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

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

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

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

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

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

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

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

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

     revision 2016-03-21 {
       description
         "Initial version. Model for NETCONF Notifications (bis)";
       reference
         "RFC XXXX: NETCONF Event Notifications";
     }


     /*
      * FEATURES
      */

     feature json {
       description
         "This feature indicates that JSON encoding of notifications
          is supported.";
     }

     feature configured-subscriptions {
       description
         "This feature indicates that management plane configuration
          of subscription is supported.";
     }

     /*
      * IDENTITIES
      */

     /* Identities for subscription results */
     identity subscription-result {
       description
         "Base identity for RPC responses to requests surrounding
          management (e.g. creation, modification) of
          subscriptions.";
     }

     identity ok {
       base subscription-result;
       description
         "OK - RPC was successful and was performed as requested.";
     }

     identity error {
       base subscription-result;
       description
         "RPC was not successful.
          Base identity for error return codes.";
     }

     identity error-no-such-subscription {
       base error;
       description
         "A subscription with the requested subscription ID
```

```
          does not exist.";
     }

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

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

     /* Identities for subscription stream status */
     identity subscription-stream-status {
       description
         "Base identity for the status of subscriptions and
          datastreams.";
     }

     identity active {
       base subscription-stream-status;
       description
         "Status is active and healthy.";
     }

     identity inactive {
       base subscription-stream-status;
       description
         "Status is inactive, for example outside the
          interval between start time and stop time.";
     }

     identity in-error {
       base subscription-stream-status;
       description
         "The status is in error or degraded, meaning that
          stream and/or subscription is currently unable to provide
          the negotiated notifications.";
     }

     /* Identities for subscription errors */
     identity subscription-errors {
       description
         "Base identity for subscription error status.
```

```
           This identity is not to be confused with error return
           codes for RPCs";
     }

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

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

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

     /* Identities for encodings */
     identity encodings {
       description
         "Base identity to represent data encodings";
     }

     identity encode-xml {
       base encodings;
       description
         "Encode data using XML";
     }

     identity encode-json {
       base encodings;
       description
         "Encode data using JSON";
     }

     /* Identities for transports */
     identity transport {
       description
         "An identity that represents a transport protocol for event
          notifications";
     }
     identity netconf {
       base transport;
       description
         "Netconf notifications as a transport.";
     }

     /*
      * TYPEDEFs
      */

     typedef subscription-id {
       type uint32;
       description
         "A type for subscription identifiers.";
     }

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

     typedef subscription-result {
       type identityref {
         base subscription-result;
       }
       description
         "The result of a subscription operation";
     }

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

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

     typedef encoding {
       type identityref {
         base encodings;
       }
       description
         "Specifies a data encoding, e.g. for a data subscription.";
     }

     typedef transport-protocol {
       type identityref {
         base transport;
       }
       description
         "Specifies transport protocol used to send notifications to a
          receiver.";
     }

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

     typedef stream-ref {
       type leafref {
         path "/netmod-notif:netconf/netmod-notif:streams/" +
               "netmod-notif:stream/netmod-notif:name";
       }
       description
         "This type is used to reference a stream.";
     }

     typedef filter-ref {
       type leafref {
         path "/notif-bis:filters/notif-bis:filter/notif-bis:filter-id";
       }
       description
```

```
          "This type is used to reference a filter.";
     }


     /*
      * GROUPINGS
      */

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

     grouping push-source-info {
       description
         "Defines the sender source from which notifications
          for a configured subscription are sent.";
       choice push-source {
         description
           "Identifies the egress interface on the Publisher from
            which notifications will or are being sent.";
         case interface-originated {
           description
             "When the push source is out of an interface on the
              Publisher established via static configuration.";
           leaf source-interface {
             type if:interface-ref;
             description
               "References the interface for notifications.";
           }
         }
         case address-originated {
           description
             "When the push source is out of an IP address on the
              Publisher established via static configuration.";
           leaf source-vrf {
             type uint32 {
               range "16..1048574";
             }
             description
               "Label of the vrf.";
           }
           leaf source-address {
             type inet:ip-address-no-zone;
             mandatory true;
             description
               "The source address for the notifications.";
           }
         }
       }
     }

     grouping receiver-info {
       description
         "Defines where and how to deliver notifications for a
          configured subscription.  This includes
          specifying the receiver, as well as defining
          any network and transport aspects when sending of
          notifications occurs outside of Netconf.";
       container receivers {
         description
           "Set of receivers in a subscription.";
         list receiver {
           key "address";
           min-elements 1;
           description
             "A single host or multipoint address intended as a target
              for the notifications for a subscription.";
           leaf address {
             type inet:host;
             mandatory true;
             description
               "Specifies the address for the traffic to reach a
                remote host. One of the following must be
                specified: an ipv4 address, an ipv6 address,
                or a host name.";
           }
           leaf port {
             type inet:port-number;
             mandatory true;
             description
               "This leaf specifies the port number to use for messages
                destined for a receiver.";
           }
           leaf protocol {
             type transport-protocol;
             default "netconf";
             description
               "This leaf specifies the transport protocol used
                to deliver messages destined for the receiver.";
           }
         }
       }
     }

     grouping subscription-response {
       description
         "Defines the output to the rpc’s establish-subscription
          and modify-subscription.";
       leaf subscription-result {
         type subscription-result;
         mandatory true;
         description
           "Indicates whether subscription is operational,
            or if a problem was encountered.";
       }
       choice result {
         description
           "Depending on the subscription result, different
            data is returned.";
         case success {
           description
             "This case is used when the subscription request
              was successful and a subscription was created/modified
              as a result";
           leaf subscription-id {
             type subscription-id;
             mandatory true;
             description
               "Identifier used for this subscription.";
           }
         }
         case no-success {
           description
             "This case applies when a subscription request
              was not successful and no subscription was
              created (or modified) as a result.  In this case,
              information MAY be returned that indicates
              suggested parameter settings that would have a
              high likelihood of succeeding in a subsequent
              establish-subscription or modify-subscription
              request.";
           uses subscription-info;
         }
       }
     }

     /*
      * RPCs
      */

     rpc establish-subscription {
       description
         "This RPC allows a subscriber to create
          (and possibly negotiate) a subscription on its own behalf.
          If successful, the subscription
          remains in effect for the duration of the subscriber's
          association with the publisher, or until the subscription
          is terminated by virtue of a delete-subscription request.
          In case an error (as indicated by subscription-result)
          is returned, the subscription is
          not created.  In that case, the RPC output
          MAY include suggested parameter settings
          that would have a high likelihood of succeeding in a
          subsequent create-subscription request.";
       input {
         uses subscription-info;
       }
       output {
         uses subscription-response;
       }
     }

     rpc modify-subscription {
       description
         "This RPC allows a subscriber to modify a subscription
```

```
          that was previously created using create-subscription.
          If successful, the subscription
          remains in effect for the duration of the subscriber’s
          association with the publisher, or until the subscription
          is terminated by virtue of a delete-subscription request.
          In case an error is returned (as indicated by
          subscription-result), the subscription is
          not modified and the original subscription parameters
          remain in effect.  In that case, the rpc error response
          MAY include suggested parameter settings
          that would have a high likelihood of succeeding in a
          subsequent modify-subscription request.";
       input {
         leaf subscription-id {
           type subscription-id;
           description
             "Identifier to use for this subscription.";
         }
         uses subscription-info;
       }
       output {
         leaf subscription-result {
           type subscription-result;
           mandatory true;
           description
             "Indicates whether subscription was modified
              or if a problem was encountered.
              In case the subscription-result has a value
              other than OK, the original subscription was not
              changed.";
         }
         uses subscription-info;
       }
     }

     rpc delete-subscription {
       description
         "This RPC allows a subscriber to delete a subscription that
          was previously created using create-subscription.";
       input {
         leaf subscription-id {
           type subscription-id;
           description
             "Identifier of the subscription that is to be deleted.
              Only subscriptions that were created using
              create-subscription can be deleted via this RPC.";
         }
       }
     }


     /*
      * NOTIFICATIONS
      */

     notification subscription-started {
       netmod-notif:control-plane-notif;
       description
         "This notification indicates that a subscription has
          started and notifications are beginning to be sent.
          This notification shall only be sent to receivers
          of a subscription; it does not constitute a general-purpose
          notification.";
       leaf subscription-id {
         type subscription-id;
         mandatory true;
         description
           "This references the affected subscription.";
       }
       uses subscription-info;
     }

     notification subscription-suspended {
       netmod-notif:control-plane-notif;
       description
         "This notification indicates that a suspension of the
          subscription by the server has occurred.  No further
          notifications will be sent until subscription
          resumes.
          This notification shall only be sent to receivers
          of a subscription; it does not constitute a general-purpose
          notification.";
       leaf subscription-id {
         type subscription-id;
         mandatory true;
         description
           "This references the affected subscription.";
       }
       leaf reason {
         type subscription-susp-reason;
         description
           "Provides a reason for why the subscription was
            suspended.";
       }
     }
     notification subscription-resumed {
       netmod-notif:control-plane-notif;
       description
         "This notification indicates that a subscription that had
          previously been suspended has resumed. Notifications
          will once again be sent.";
       leaf subscription-id {
         type subscription-id;
         mandatory true;
         description
           "This references the affected subscription.";
       }
     }

     notification subscription-modified {
       netmod-notif:control-plane-notif;
       description
         "This notification indicates that a subscription has
          been modified.  Notifications sent from this point
          on will conform to the modified terms of the
          subscription.";
       leaf subscription-id {
         type subscription-id;
         mandatory true;
         description
           "This references the affected subscription.";
       }
       uses subscription-info;
     }

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

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

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

     augment /netmod-notif:notificationComplete {
       description
```

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




     /*
      * DATA NODES
      */

     container filters {
       description
         "This container contains a list of configurable filters
          that can be applied to subscriptions.  This facilitates
          the reuse of complex filters once defined.";
       list filter {
         key "filter-id";
         description
           "A list of configurable filters that can be applied to
            subscriptions.";
         leaf filter-id {
           type filter-id;
           description
             "An identifier to differentiate between filters.";
         }
         uses notif:base-filter;
       }
     }
     container subscription-config {
       if-feature "configured-subscriptions";
       description
         "Contains the list of subscriptions that are configured,
          as opposed to established via RPC or other means.";
       list subscription {
         key "subscription-id";
         description
           "Content of a subscription.";
         leaf subscription-id {
           type subscription-id;
           description
```

```
              "Identifier to use for this subscription.";
          }
          uses subscription-info;
          uses receiver-info {
            if-feature "configured-subscriptions";
          }
          uses push-source-info {
            if-feature "configured-subscriptions";
          }
        }
      }
      container subscriptions {
        config false;
        description
          "Contains the list of currently active subscriptions,
           i.e. subscriptions that are currently in effect,
           used for subscription management and monitoring purposes.
           This includes subscriptions that have been setup via RPC
           primitives, e.g. create-subscription, delete-subscription,
           and modify-subscription, as well as subscriptions that
           have been established via configuration.";
        list subscriptions {
          key "subscription-id";
          config false;
          description
            "Content of a subscription.
             Subscriptions can be created using a control channel
             or RPC, or be established through configuration.";
          leaf subscription-id {
            type subscription-id;
            description
              "Identifier of this subscription.";
          }
          choice subscription-type {
            description
              "This choice contains a set of flags to indicate
               the type of subscription.";
            case via-configuration {
              leaf configured-subscription {
                if-feature "configured-subscriptions";
                type empty;
                description
                  "The presence of this leaf indicates that the
                   subscription originated from configuration, not
                   through a control channel or RPC.";
              }
            }
            case created-via-RPC {
              leaf created-subscription {
                type empty;
                description
                  "The presence of this leaf indicates that the
                   subscription originated via RPC using RFC5277
                   mechanisms and semantics.";
              }
            }
          }
          leaf subscription-status {
            type identityref {
              base subscription-stream-status;
            }
            description
              "The status of the subscription.";
          }
          uses subscription-info;
          uses receiver-info {
            if-feature "configured-subscriptions";
          }
          uses push-source-info {
            if-feature "configured-subscriptions";
          }
        }
      }

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

    }

    <CODE ENDS>
```

4. Backwards Compatibility

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

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

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

Figure 23: Hello message

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

Note that to support backwards compatibility, the yang models in this document include two types of naming conventions. That used in [RFC5277], e.g., replayComplete; and that commonly used in yang models, e.g., subscription-started.

5. Security Considerations

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

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

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

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

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

A subscription could be configured on another receiver’s behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver, which doesn’t even support subscriptions. Clients that do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, the NETCONF Authorization Control Model [RFC6536] SHOULD be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user permissions to receive specific event notification types. The permissions are specified as a set of access control rules.

Note that streams can define additional authorization requirements. For instance, in [netconf-yang-push] each of the elements in its data plane notifications must also go through access control.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

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

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

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

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

6.2. Informative References

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

Authors’ Addresses

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Alexander Clemm
Cisco Systems

Email: alex@cisco.com

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

NETCONF Working Group   K. Watsen
Internet-Draft   Juniper Networks
Intended status: Standards Track   J. Schoenwaelder
Expires: September 17, 2016   Jacobs University Bremen
                                                          March 16, 2016

NETCONF Server and RESTCONF Server Configuration Models
draft-ietf-netconf-server-model-09

Abstract

This draft defines a NETCONF server configuration data model and a RESTCONF server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listen on, call-home parameters, client authentication, and related parameters.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. Please note that no other RFC Editor instructions are specified anywhere else in this document.

This document contains references to other drafts in progress, both in the Normative References section, as well as in body text throughout. Please update the following references to reflect their final RFC assignments:

- draft-ietf-netconf-restconf
- draft-ietf-netconf-call-home
- draft-ietf-rtgwg-yang-key-chain

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- "VVVV" --> the assigned RFC value for this draft
- "XXXX" --> the assigned RFC value for draft-ietf-netconf-restconf
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-call-home

Artwork in this document contains placeholder values for ports pending IANA assignment from "draft-ietf-netconf-call-home". Please apply the following replacements:

- "7777" --> the assigned port value for "netconf-ch-ssh"
- "8888" --> the assigned port value for "netconf-ch-tls"
- "9999" --> the assigned port value for "restconf-ch-tls"

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- "2016-03-16" --> the publication date of this draft

The following two Appendix sections are to be removed prior to publication:

- Appendix A. Change Log
- Appendix B. Open Issues

Artwork in the document contains a temporary YANG containers that need to be removed.

- The "listening-ssh-server" container listed at the end of the artwork in Section 4.2.3 needs to be removed. Please remove the ten lines starting with "container listening-ssh-server {" and ending with "}".
- The "listening-tls-server" container listed at the end of the artwork in Section 4.3.3 needs to be removed. Please remove the ten lines starting with "container listening-tls-server {" and ending with "}".

Status of This Memo

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

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

Copyright Notice

Table of Contents

1. Introduction

This draft defines a NETCONF [RFC6241] server configuration data model and a RESTCONF [draft-ietf-netconf-restconf] server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listen on, call-home parameters, client authentication, and related parameters.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Tree Diagrams

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- Brackets "[" and "]" enclose list keys.
- Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Objectives

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF and RESTCONF services on a network element. This scope includes the following objectives:

2.1. Support all NETCONF and RESTCONF transports

The YANG module should support all current NETCONF and RESTCONF transports, namely NETCONF over SSH [RFC6242], NETCONF over TLS [RFC7589], and RESTCONF over TLS [draft-ietf-netconf-restconf], and to be extensible to support future transports as necessary.

Because implementations may not support all transports, the module should use YANG "feature" statements so that implementations can accurately advertise which transports are supported.

2.2. Enable each transport to select which keys to use

Servers may have a multiplicity of host-keys or server-certificates from which subsets may be selected for specific uses. For instance, a NETCONF server may want to use one set of SSH host-keys when listening on port 830, and a different set of SSH host-keys when calling home. The data models provided herein should enable configuration of which keys to use on a per-use basis.

2.3. Support authenticating NETCONF/RESTCONF clients certificates

When a certificate is used to authenticate a NETCONF or RESTCONF client, there is a need to configure the server to know how to authenticate the certificates. The server should be able to authenticate the client’s certificate either by using path-validation to a configured trust anchor or by matching the client-certificate to one previously configured.

2.4. Support mapping authenticated NETCONF/RESTCONF client certificates to usernames

When a client certificate is used for TLS client authentication, the NETCONF/RESTCONF server must be able to derive a username from the authenticated certificate. Thus the modules defined herein should enable this mapping to be configured.

2.5. Support both listening for connections and call home

The NETCONF and RESTCONF protocols were originally defined as having the server opening a port to listen for client connections. More recently the NETCONF working group defined support for call-home ([draft-ietf-netconf-call-home]), enabling the server to initiate the connection to the client, for both the NETCONF and RESTCONF protocols. Thus the modules defined herein should enable configuration for both listening for connections and calling home. Because implementations may not support both listening for connections and calling home, YANG "feature" statements should be used so that implementation can accurately advertise the connection types it supports.

2.6. For Call Home connections

The following objectives only pertain to call home connections.

2.6.1. Support more than one NETCONF/RESTCONF client

A NETCONF/RESTCONF server may be managed by more than one NETCONF/RESTCONF client. For instance, a deployment may have one client for provisioning and another for fault monitoring. Therefore, when it is desired for a server to initiate call home connections, it should be able to do so to more than one client.

2.6.2. Support NETCONF/RESTCONF clients having more than one endpoint

An NETCONF/RESTCONF client managing a NETCONF/RESTCONF server may implement a high-availability strategy employing a multiplicity of active and/or passive endpoint. Therefore, when it is desired for a server to initiate call home connections, it should be able to connect to any of the client’s endpoints.

2.6.3. Support a reconnection strategy

Assuming a NETCONF/RESTCONF client has more than one endpoint, then it becomes necessary to configure how a NETCONF/RESTCONF server should reconnect to the client should it lose its connection to one the client’s endpoints. For instance, the NETCONF/RESTCONF server may start with first endpoint defined in a user-ordered list of endpoints or with the last endpoints it was connected to.

2.6.4. Support both persistent and periodic connections

NETCONF/RESTCONF clients may vary greatly on how frequently they need to interact with a NETCONF/RESTCONF server, how responsive interactions need to be, and how many simultaneous connections they can support. Some clients may need a persistent connection to servers to optimize real-time interactions, while others prefer periodic interactions in order to minimize resource requirements. Therefore, when it is necessary for server to initiate connections, it should be configurable if the connection is persistent or periodic.

2.6.5. Reconnection strategy for periodic connections

The reconnection strategy should apply to both persistent and periodic connections. How it applies to periodic connections becomes clear when considering that a periodic "connection" is a logical connection to a single server. That is, the periods of unconnectedness are intentional as opposed to due to external reasons. A periodic "connection" should always reconnect to the same server until it is no longer able to, at which time the reconnection strategy guides how to connect to another server.

2.6.6. Keep-alives for persistent connections

If a persistent connection is desired, it is the responsibility of the connection initiator to actively test the "aliveness" of the connection. The connection initiator must immediately work to reestablish a persistent connection as soon as the connection is lost. How often the connection should be tested is driven by NETCONF/RESTCONF client requirements, and therefore keep-alive settings should be configurable on a per-client basis.

2.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the NETCONF/RESTCONF server to know how often it should connect. This frequency determines the maximum amount of time a NETCONF/RESTCONF client may have to wait to send data to a server. A server may connect to a client before this interval expires if desired (e.g., to send data to a client).

3. High-Level Design

The solution presented in this document defines a configurable keychain object, reusable groupings for SSH and TLS based servers, and, finally, the configurable NETCONF and RESTCONF server objects, which are the primary purpose for this draft. Each of these are defined in a distinct YANG module, thus a total of five YANG modules are defined in this document. The relationship between these five YANG modules is illustrated by the tree diagram below.

```
                  +--------------------+
                  |ietf-system-keychain|
                  +--------------------+
                         ^    ^
                         |    |
              <leafref>  |    |  <leafref>
            +------------+    +------------+
            |                              |
     +---------------+            +------------------+
     |ietf-ssh-server|            | ietf-tls-server  |
     +---------------+            +------------------+
       ^                               ^       ^
       | <uses>                        |       |
       |            <augments>         |       |
       |          +--------------------+       | <augments>
       |          |                            |
   +-------------------+         +--------------------+
   |ietf-netconf-server|         |ietf-restconf-server|
   +-------------------+         +--------------------+
```

4. Solution

Each of the following five sections relate to one of the YANG modules depicted by the figure above.

4.1. The System Keychain Model

The system keychain model defined in this section provides a configurable object having the following characteristics:

- A semi-configurable list of private keys, each with one or more associated certificates. Private keys MUST be either preinstalled (e.g., an IDevID key), be generated by request, or be loaded by request. Each private key is MAY have associated certificates, either preinstalled or configured after creation.

- A configurable list of lists of trust anchor certificates. This enables the server to have use-specific trust anchors. For instance, one list of trust anchors might be used to authenticate management connections (e.g., client certificate-based authentication for NETCONF or RESTCONF connections), and a different list of trust anchors might be used for when connecting to a specific Internet-based service (e.g., a zero touch bootstrap server).

- An RPC to generate a certificate signing request for an existing private key, a passed subject, and an optional attributes. The signed certificate returned from an external certificate authority (CA) can be later set using a standard configuration change request (e.g., <edit-config>).

- An RPC to request the server to generate a new private key using the specified algorithm and key length.

- An RPC to request the server to load a new private key.

4.1.1. Tree Diagram

```
 module: ietf-system-keychain
    +--rw keychain
       +--rw private-keys
       |  +--rw private-key* [name]
       |  |  +--rw name                                    string
       |  |  +--ro algorithm?                              kc:algorithms
       |  |  +--ro key-length?                             uint32
       |  |  +--ro public-key                              binary
       |  |  +--rw certificate-chains
       |  |  |  +--rw certificate-chain* [name]
       |  |  |     +--rw name           string
       |  |  |     +--rw certificate*   binary
       |  |  +---x generate-certificate-signing-request
       |  |     +---w input
       |  |     |  +---w subject       binary
       |  |     |  +---w attributes?   binary
       |  |     +--ro output
       |  |        +--ro certificate-signing-request    binary
       |  +---x generate-private-key
       |  |  +---w input
       |  |     +---w name          string
       |  |     +---w key-usage?    enumeration
       |  |     +---w algorithm     kc:algorithms
       |  |     +---w key-length?   uint32
       |  +---x load-private-key
       |     +---w input
       |        +---w name           string
       |        +---w private-key    binary
       +--rw trusted-certificates* [name]
          +--rw name                   string
          +--rw description?           string
          +--rw trusted-certificate* [name]
             +--rw name           string
             +--rw certificate?   binary
 notifications:
    +---n certificate-expiration
       +--ro certificate       instance-identifier
       +--ro expiration-date    yang:date-and-time
```

4.1.2. Example Usage

The following example illustrates the "generate-private-key" action in use with the RESTCONF protocol and JSON encoding.

```
REQUEST
-------

  ['\' line wrapping added for formatting only]

  POST https://example.com/restconf/data/ietf-system-keychain:keychain/\
  private-keys/generate-private-key HTTP/1.1
  HOST: example.com
  Content-Type: application/yang.operation+json

  {
    "ietf-system-keychain:input" : {
      "name" : "ex-key-sect571r1",
      "algorithm" : "sect571r1"
    }
  }


RESPONSE
--------

  HTTP/1.1 204 No Content
  Date: Mon, 31 Oct 2015 11:01:00 GMT
  Server: example-server
```

The following example illustrates the "load-private-key" action in use with the RESTCONF protocol and JSON encoding.

```
REQUEST
-------

  ['\' line wrapping added for formatting only]

  POST https://example.com/restconf/data/ietf-system-keychain:keychain/\
  private-keys/generate-private-key HTTP/1.1
  HOST: example.com
  Content-Type: application/yang.operation+xml

  <input xmlns="urn:ietf:params:xml:ns:yang:ietf-system-keychain">
    <name>ex-key-sect571r1</name>
    <private-key>
      NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
      VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
      V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
      Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
      QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
      MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
      NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
      WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
      lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
      zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
      25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
      WpiMjB2WlhoaGJYQnNaUzVqY215aU9L=
    </private-key>
  </input>


RESPONSE
--------

  HTTP/1.1 204 No Content
  Date: Mon, 31 Oct 2015 11:01:00 GMT
  Server: example-server
```

The following example illustrates the "generate-certificate-signing-request" action in use with the NETCONF protocol.

```
REQUEST
-------

  <rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <action xmlns="urn:ietf:params:xml:ns:yang:1">
      <keychain
        xmlns="urn:ietf:params:xml:ns:yang:ietf-system-keychain">
```

```
           <private-keys>
             <private-key>
               <name>ex-key-sect571r1</name>
               <generate-certificate-signing-request>
                 <subject>
                  cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2R
                  manZvO3NkZmJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNlmO
                  Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmR6Zgo=
                 </subject>
                 <attributes>
                  bwtakWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvut4
                  arnZvO3NkZmJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNkYm
                  Z2aXNiZGZpYmhzZG87ZmJvO3NkZ25iO29pLmC6Rhp=
                 </attributes>
               </generate-certificate-signing-request>
             </private-key>
           </private-keys>
         </keychain>
       </action>
     </rpc>


   RESPONSE
   --------

     <rpc-reply message-id="101"
        xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <certificate-signing-request
          xmlns="urn:ietf:params:xml:ns:yang:ietf-system-keychain">
          LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
          0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
          FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
          GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
          diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
          KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
          El1QWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJTdGZQY3N0Zk1KT1
          FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAvU25FcFE0TnV
          bXBDT2YKQWdNQkFBR2pnYXd3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
          URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTFVZApJd1JkTUZ1QU
          ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
          mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
          RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
          rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
          TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
          c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
          SWHgzZjdVM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
        </certificate-signing-request>
     </rpc-reply>
```

The following example illustrates what a fully configured keychain object might look like. The private-key shown below is consistent with the generate-private-key and generate-certificate-signing-request examples above. This example also assumes that the resulting CA-signed certificate has been configured back onto the server. Lastly, this example shows that three lists of trusted certificates having been configured.

```
<keychain xmlns="urn:ietf:params:xml:ns:yang:ietf-system-keychain">

  <!-- private keys and associated certificates -->
  <private-keys>
    <private-key>
      <name>tpm-protected-key</name>
      <algorithm>sect571r1</algorithm>
      <public-key>
        cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
        mJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNkYmZ2aXNiZGZpYmhzZG87Zm
        JvO3NkZ25iO29pLmR6Zgo=
      </public-key>
      <certificate-chains>
        <certificate-chain>
          <name>default-idevid-chain</name>
          <certificate>
            diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
            LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
            KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
            0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
            FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
            GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
            ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
            mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
            RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
            rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
            TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
            c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
            SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
          </certificate>
          <certificate>
            KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
            El1QWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJTdGZQY3N0Zk1KT1
            FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAvU25FcFE0TnV
            bXBDT2YKQWdNQkFBR2pnYXd3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
            LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
            0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
            FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
            GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
            diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
```

```
               URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTFVZApJd1JkTUZ1QU
               RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
               rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
               c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
               SSUZJQ0FURS0tLS0tCg==
             </certificate>
           </certificate-chain>
           <certificate-chain>
             <name>my-ldevid-chain</name>
             <certificate>
               0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
               FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
               GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
               diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
               LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
               KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
               El1QWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJTdGZQY3N0Zk1KT1
               FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAvU25FcFE0TnV
               ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
               mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
               RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
               rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
               TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
               c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
               SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
             </certificate>
             <certificate>
               LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
               0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
               FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
               GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
               diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
               KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
               El1QWMrQ1RsTkNmc0d6cEw1Um5ydXZsOFRIcUJTdGZQY3N0Zk1KT1
               FaNzlnNlNWVldsMldzaHE1bUViCkJNNitGNzdjbTAvU25FcFE0TnV
               bXBDT2YKQWdNQkFBR2pnYXd3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
               URiR0lPNDB4ajlPb3JtREdsRUNCVTFNR1FHQTFVZApJd1JkTUZ1QU
               ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
               mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
               RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
               rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
               TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
               c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
               SWHgzZjdVM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
             </certificate>
           </certificate-chain>
         </certificate-chains>
       </private-key>
```

```
     </private-keys>

     <!-- trusted netconf/restconf client certificates -->
     <trusted-certificates>
       <name>explicitly-trusted-client-certs</name>
       <description>
         Specific client authentication certificates that are to be
         explicitly trusted NETCONF/RESTCONF clients.  These are
         needed for client certificates not signed by our CA.
       </description>
       <trusted-certificate>
         <name>George Jetson</name>
         <certificate>
           QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ
           MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
           25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2
           RV0JCU2t2MXI2SFNHeUFUVkpwSmYyOWtXbUU0NEo5akJrQmdOVkhTTUVY
           VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
           UxNQWtHQTFVRUJoTUNWVk14RURBT0JnTlZCQW9UQjJWNApZVzF3YkdVeE
           V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF
           NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
           Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGGKYlhCc1pTN
           WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
           xWVE1SQXdEZ1lEVlFRSwpFd2RsZUdGdGNHeGxNUk13RVFZRFZRUURFd3B
           EVWt3Z1NYTnpkV1Z5TUEwR0NTcUdTSWIzRFFFQkJRVUFBNEdCCkFFc3BK
           WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
           TQzcjFZSjk0M1FQLzV5eGUKN2QxMkxCV0dxUjUrbEl5N01YL21ka2M4al
           zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
           LS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
         </certificate>
       </trusted-certificate>
       <trusted-certificate>
         <name>Fred Flintstone</name>
         <certificate>
           VlEVlFRREV3Vm9ZWEJ3ZVRDQm56QU5CZ2txaGtpRzl3MEJBUUVGQUFPQm
           pRQXdnWWtDCmdZRUE1RzRFSWZsS1p2bDlXTW44eUhyM2hObUFRaUhVUzV
           rRUpPQy9hSFA3eGJXQW1ra054ZStUa2hrZnBsL3UKbVhsTjhSZUd1ODhG
           NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd
           VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
           V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF
           NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
           Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGGKYlhCc1pTN
           WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
           xWVE1SQXdEZ1lEVlFRSwpFd2RsZUdGdGNHeGxNUk13RVFZRFZRUURFd3B
           EVWt3Z1NYTnpkV1Z5TUEwR0NTcUdTSWIzRFFFQkJRVUFBNEdCCkFFc3BK
           WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
           lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk
           zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
           QWtUOCBDRVUUZJ0RUF==
         </certificate>
       </trusted-certificate>
     </trusted-certificates>

     <!-- trust anchors for netconf/restconf clients -->
     <trusted-certificates>
       <name>deployment-specific-ca-certs</name>
       <description>
         Trust anchors used only to authenticate NETCONF/RESTCONF
         client connections.  Since our security policy only allows
         authentication for clients having a certificate signed by
         our CA, we only configure its certificate below.
       </description>
       <trusted-certificate>
         <name>ca.example.com</name>
         <certificate>
           WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
           lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk
           zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
           NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd
           VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
           V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF
           NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
           Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN
           WpiMjB2WlhoaGJJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
           QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbApLR0Z0Y0d4bE1RNHdEQ
           MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
           25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2
           RJSUJQFRStS0Cg==
         </certificate>
       </trusted-certificate>
     </trusted-certificates>

     <!-- trust anchors for random HTTPS servers on Internet -->
     <trusted-certificates>
       <name>common-ca-certs</name>
       <description>
         Trusted certificates to authenticate common HTTPS servers.
         These certificates are similar to those that might be
         shipped with a web browser.
       </description>
       <trusted-certificate>
         <name>ex-certificate-authority</name>
         <certificate>
           NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd
           VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
           V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF
           Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN
           QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ
           MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ
           NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC
           WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM
           lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk
           zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot
           25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2
           WpiMjB2WlhoaGJYQnNaUzVqY215aU9L=
         </certificate>
       </trusted-certificate>
     </trusted-certificates>

   </keychain>
```

The following example illustrates a "certificate-expiration" notification in XML.

```
['\' line wrapping added for formatting only]

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-07-08T00:01:00Z</eventTime>
  <certificate-expiration
    xmlns="urn:ietf:params:xml:ns:yang:ietf-system-keychain">
    <certificate>
      /kc:keychain/kc:private-keys/kc:private-key/kc:certificate-chains\
      /kc:certificate-chain/kc:certificate[3]
    </certificate>
    <expiration-date>2016-08-08T14:18:53-05:00</expiration-date>
  </certificate-expiration>
</notification>
```

4.1.3. YANG Model

This YANG module makes extensive use of data types defined in [RFC5280] and [RFC5958].

```
<CODE BEGINS> file "ietf-system-keychain@2016-03-16.yang"

module ietf-system-keychain {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-system-keychain";
  prefix "kc";

  import ietf-yang-types {     // RFC 6991
```

```
    prefix yang;
  }

  organization
   "IETF NETCONF (Network Configuration) Working Group";

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

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

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

    Editor:   Kent Watsen
              <mailto:kwatsen@juniper.net>";


  description
   "This module defines a keychain to centralize management of
    security credentials.

    Copyright (c) 2014 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
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC VVVV; see
    the RFC itself for full legal notices.";

  revision "2016-03-16" {
    description
     "Initial version";
    reference
     "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                Models";
  }

  typedef algorithms {
    type enumeration {
      enum rsa { description "The RSA algorithm."; }
```

```
        enum secp192r1 { description "The secp192r1 algorithm."; }
        enum secp256r1 { description "The secp256r1 algorithm."; }
        enum secp384r1 { description "The secp384r1 algorithm."; }
        enum secp521r1 { description "The secp521r1 algorithm."; }
        // what about ecdh_x25519 and ecdh_x448 in TLS 1.3?
      }
      description
        "Asymmetric key algorithms.  This list has been trimmed down
         to the minimal subset of algorithms recommended by the IETF.
         Please see the Design Consideration section in RFC VVVV for
         more information about this.";
    }

    container keychain {
      description
        "A list of private-keys and their associated certificates, as
         well as lists of trusted certificates for client certificate
         authentication.  RPCs are provided to generate a new private
         key and to generate a certificate signing requests.";

      container private-keys {
        description
          "A list of private key maintained by the keychain.";
        list private-key {
          key name;
          description
            "A private key.";
          leaf name {
            type string;
            description
              "An arbitrary name for the private key.";
          }
          leaf algorithm {
            type kc:algorithms;
            config false;
            description
              "The algorithm used by the private key.";
          }
          leaf key-length {
            type uint32;
            config false;
            description
              "The key-length used by the private key.";
          }
          leaf public-key {
            type binary;
            config false;
            mandatory true;
            description
              "An OneAsymmetricKey ’publicKey’ structure as specified
               by RFC 5958, Section 2 encoded using the ASN.1
               distinguished encoding rules (DER), as specified
               in ITU-T X.690.";
            reference
              "RFC 5958:
                  Asymmetric Key Packages
               ITU-T X.690:
                  Information technology - ASN.1 encoding rules:
                  Specification of Basic Encoding Rules (BER),
                  Canonical Encoding Rules (CER) and Distinguished
                  Encoding Rules (DER).";
          }
          container certificate-chains {
            description
              "Certificate chains associated with this private key.
               More than one chain per key is enabled to support,
               for instance, a TPM-protected key that has associated
               both IDevID and LDevID certificates.";
            list certificate-chain {
              key name;
              description
                "A certificate chain for this public key.";
              leaf name {
                type string;
                description
                  "An arbitrary name for the certificate chain.";
              }
              leaf-list certificate {
                type binary;
                ordered-by user;
                description
                  "An X.509 v3 certificate structure as specified by RFC
                   5280, Section 4 encoded using the ASN.1 distinguished
                   encoding rules (DER), as specified in ITU-T X.690.
                   The list of certificates that run from the server
                   certificate towards the trust anchor.  The chain MAY
                   include the trust anchor certificate itself.";
                reference
                  "RFC 5280:
                     Internet X.509 Public Key Infrastructure Certificate
                     and Certificate Revocation List (CRL) Profile.
                   ITU-T X.690:
                      Information technology - ASN.1 encoding rules:
                      Specification of Basic Encoding Rules (BER),
                      Canonical Encoding Rules (CER) and Distinguished
                      Encoding Rules (DER).";
              }
            }
          }
          action generate-certificate-signing-request {
            description
              "Generates a certificate signing request structure for
               the associated private key using the passed subject and
               attribute values.  Please review both the Security
               Considerations and Design Considerations sections in
               RFC VVVV for more information regarding this action
               statement.";
            input {
              leaf subject {
                type binary;
                mandatory true;
                description
                  "The 'subject' field from the CertificationRequestInfo
                   structure as specified by RFC 2986, Section 4.1 encoded
                   using the ASN.1 distinguished encoding rules (DER), as
                   specified in ITU-T X.690.";
                reference
                  "RFC 2986:
                     PKCS #10: Certification Request Syntax Specification
                     Version 1.7.
                   ITU-T X.690:
                      Information technology - ASN.1 encoding rules:
                      Specification of Basic Encoding Rules (BER),
                      Canonical Encoding Rules (CER) and Distinguished
                      Encoding Rules (DER).";
              }
              leaf attributes {
                type binary;
                description
                 "The 'attributes' field from the CertificationRequestInfo
                  structure as specified by RFC 2986, Section 4.1 encoded
                  using the ASN.1 distinguished encoding rules (DER), as
                  specified in ITU-T X.690.";
                reference
                  "RFC 2986:
                     PKCS #10: Certification Request Syntax Specification
                     Version 1.7.
                   ITU-T X.690:
                      Information technology - ASN.1 encoding rules:
                      Specification of Basic Encoding Rules (BER),
                      Canonical Encoding Rules (CER) and Distinguished
                      Encoding Rules (DER).";
              }
            }
            output {
              leaf certificate-signing-request {
                type binary;
                mandatory true;
                description
                  "A CertificationRequest structure as specified by RFC
                   2986, Section 4.1 encoded using the ASN.1 distinguished
                   encoding rules (DER), as specified in ITU-T X.690.";
                reference
                  "RFC 2986:
                     PKCS #10: Certification Request Syntax Specification
                     Version 1.7.
                   ITU-T X.690:
                      Information technology - ASN.1 encoding rules:
                      Specification of Basic Encoding Rules (BER),
                      Canonical Encoding Rules (CER) and Distinguished
                      Encoding Rules (DER).";

              }
            }
          }
        }

        action generate-private-key {
          description
            "Requests the device to generate a private key using the
             specified algorithm and key length.";
          input {
            leaf name {
              type string;
              mandatory true;
              description
                "The name this private-key should have when listed
                 in /keychain/private-keys.  As such, the passed
                 value must not match any existing ’name’ value.";
            }
            leaf key-usage {
              type enumeration {
                enum signing    { description "signing"; }
                enum encryption { description "encryption"; }
                // unclear if these should be somehow more
                // specific or varied.
              }
              description
                "An optional parameter further restricting the use of
                 this key.  Some algorithms inherently restrict use
                 (DH for signing) whereas others can support more than
                 one use (RSA).  This flag forces the device to only
                 allow the key to be used for the indicated purposes.";
            }
            leaf algorithm {
              type kc:algorithms;
              mandatory true;
              description
                "The algorithm to be used when generating the key.";
            }
            leaf key-length {
              type uint32;
              description
                "For algorithms that need a key length specified
                 when generating the key.";
            }
          }
        }

        action load-private-key {
          description
            "Requests the device to load a private key";
          input {
            leaf name {
              type string;
              mandatory true;
              description
                "The name this private-key should have when listed
                 in /keychain/private-keys.  As such, the passed
                 value must not match any existing ’name’ value.";
            }
            leaf private-key {
              type binary;
              mandatory true;
                description
                  "An OneAsymmetricKey structure as specified by RFC
                   5958, Section 2 encoded using the ASN.1 distinguished
                   encoding rules (DER), as specified in ITU-T X.690.
                   Note that this is the raw private with no shrouding
                   to protect it.  The strength of this private key
                   MUST NOT be greater than the strength of the secure
                   connection over which it is communicated.  Devices
                   SHOULD fail this request if ever that happens.";
                reference
                  "RFC 5958:
                      Asymmetric Key Packages
                   ITU-T X.690:
                      Information technology - ASN.1 encoding rules:
                      Specification of Basic Encoding Rules (BER),
                      Canonical Encoding Rules (CER) and Distinguished
                      Encoding Rules (DER).";
            }
          }
        }
      }

      list trusted-certificates {
        key name;
        description
          "A list of trusted certificates.  Each list SHOULD be specific
           to a purpose.  For instance, there could be one list for
           authenticating NETCONF/RESTCONF client certificates, and
           another list for authenticating manufacturer-signed data,
           and yet another list for authenticated web servers.";
        leaf name {
          type string;
          description
            "An arbitrary name for this list of trusted certificates.";
        }
        leaf description {
          type string;
          description
            "An arbitrary description for this list of trusted
             certificates.";
        }
        list trusted-certificate {
          key name;
          description
            "A trusted certificate for a specific use.";
          leaf name {
            type string;
            description
              "An arbitrary name for this trusted certificate.";
          }
          leaf certificate {
            type binary;
            description
              "An X.509 v3 certificate structure as specified by RFC
               5280, Section 4 encoded using the ASN.1 distinguished
               encoding rules (DER), as specified in ITU-T X.690.";
            reference
              "RFC 5280:
                 Internet X.509 Public Key Infrastructure Certificate
                 and Certificate Revocation List (CRL) Profile.
               ITU-T X.690:
                  Information technology - ASN.1 encoding rules:
                  Specification of Basic Encoding Rules (BER),
                  Canonical Encoding Rules (CER) and Distinguished
                  Encoding Rules (DER).";
          }
        }
      }
    }
    notification certificate-expiration {
      description
        "A notification indicating that a configured certificate is
         either about to expire or has already expired.  When to send
         notifications is an implementation specific decision, but
         it is RECOMMENDED that a notification be sent once a month
         for 3 months, then once a week for four weeks, and then once
         a day thereafter.";
      leaf certificate {
        type instance-identifier;
        mandatory true;
        description
          "Identifies which certificate is expiring or is expired.";
      }
      leaf expiration-date {
        type yang:date-and-time;
        mandatory true;
        description
          "Identifies the expiration date on the certificate.";
      }
    }
  }
```

<CODE ENDS>

4.2. The SSH Server Model

The SSH Server model presented in this section presents two YANG groupings, one for a server that opens a socket to accept TCP connections on, and another for a server that has had the TCP connection opened for it already (e.g., inetd).

The SSH Server model (like the TLS Server model presented below) is provided as a grouping so that it can be used in different contexts. For instance, the NETCONF Server model presented in Section 4.4 uses one grouping to configure a NETCONF server listening for connections and the other grouping to configure NETCONF call home.

A shared characteristic between both groupings is the ability to configure which host key is presented to clients, the private key for which is held in the keychain configuration presented before. Another shared characteristic is the ability to configure which trusted CA or client certificates the server should be used to authenticate clients when using X.509 based client certificates [RFC6187].

4.2.1. Tree Diagram

The following tree diagram represents the data model for the grouping used to configure an SSH server to listen for TCP connections. The tree diagram for the other grouping is not provided, but it is the same except without the "address" and "port" fields.

NOTE: the diagram below shows "listening-ssh-server" as a YANG container (not a grouping). This temporary container was created only to enable the ‘pyang‘ tool to output the tree diagram, as groupings by themselves have no protocol accessible nodes, and hence ‘pyang‘ would output an empty tree diagram.

```
module: ietf-ssh-server
   +--rw listening-ssh-server
      +--rw address?            inet:ip-address
      +--rw port                inet:port-number
      +--rw host-keys
      |  +--rw host-key* [name]
      |     +--rw name           string
      |     +--rw (type)?
      |        +--:(public-key)
      |        |  +--rw public-key?    -> /kc:keychain/private-keys/pri
vate-key/name
      |        +--:(certificate)
      |           +--rw certificate?   -> /kc:keychain/private-keys/pri
vate-key/certificate-chains/certificate-chain/certificate {ssh-x509-cer
ts}?
      +--rw client-cert-auth {ssh-x509-certs}?
         +--rw trusted-ca-certs?       -> /kc:keychain/trusted-certific
ates/name
         +--rw trusted-client-certs?   -> /kc:keychain/trusted-certific
ates/name
```

4.2.2. Example Usage

This section shows how it would appear if the temporary listening-ssh-server container just mentioned above were populated with some data. This example is consistent with the examples presented earlier in this document.

```
   <listening-ssh-server
     xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">
     <port>830</port>
     <host-keys>
       <host-key>
         <name>deployment-specific-certificate</name>
         <certificate>ex-key-sect571r1-cert</certificate>
       </host-key>
     </host-keys>
     </certificates>
     <client-cert-auth>
       <trusted-ca-certs>
         deployment-specific-ca-certs
       </trusted-ca-certs>
       <trusted-client-certs>
         explicitly-trusted-client-certs
       </trusted-client-certs>
     </client-cert-auth>
   </listening-ssh-server>
```

4.2.3. YANG Model

This YANG module has a normative reference to [RFC4253].

```
  <CODE BEGINS> file "ietf-ssh-server@2016-03-16.yang"

  module ietf-ssh-server {
    yang-version 1.1;

    namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
    prefix "ts";

    import ietf-inet-types {           // RFC 6991
      prefix inet;
    }
    import ietf-system-keychain {
      prefix kc;                       // RFC VVVV
      revision-date 2016-03-16;
    }

    organization
     "IETF NETCONF (Network Configuration) Working Group";

    contact
     "WG Web:   <http://tools.ietf.org/wg/netconf/>
      WG List:  <mailto:netconf@ietf.org>
      WG Chair: Mehmet Ersue
                <mailto:mehmet.ersue@nsn.com>

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

      Editor:   Kent Watsen
                <mailto:kwatsen@juniper.net>";


    description
     "This module defines a reusable grouping for a SSH server that
      can be used as a basis for specific SSH server instances.

      Copyright (c) 2014 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
      (http://trustee.ietf.org/license-info).

      This version of this YANG module is part of RFC VVVV; see
      the RFC itself for full legal notices.";

    revision "2016-03-16" {
      description
       "Initial version";
      reference
       "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                  Models";
    }

    // features
    feature ssh-x509-certs {
      description
        "The ssh-x509-certs feature indicates that the NETCONF
         server supports RFC 6187";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
         Authentication";
    }

    // grouping
    grouping non-listening-ssh-server-grouping {
      description
```

```
        "A reusable grouping for a SSH server that can be used as a
         basis for specific SSH server instances.";

      container host-keys {
        description
          "The list of host-keys the SSH server will present when
           establishing a SSH connection.";
        list host-key {
          key name;
          min-elements 1;
          ordered-by user;
          description
            "An ordered list of host keys the SSH server will use to
             construct its ordered list of algorithms, when sending
             its SSH_MSG_KEXINIT message, as defined in Section 7.1
             of RFC 4253.";
          reference
            "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
          leaf name {
            type string;
            mandatory true;
            description
              "An arbitrary name for this host-key";
          }
          choice type {
            description
              "The type of host key being specified";
            leaf public-key {
              type leafref {
                path "/kc:keychain/kc:private-keys/kc:private-key/"
                     + "kc:name";
              }
              description
                "The public key is actually identified by the name of
                 its cooresponding private-key in the keychain.";
            }
            leaf certificate {
              if-feature ssh-x509-certs;
              type leafref {
                path "/kc:keychain/kc:private-keys/kc:private-key/"
                     + "kc:certificate-chains/kc:certificate-chain/"
                     + "kc:certificate";
              }
              description
                "The name of a certificate in the keychain.";
            }
          }
        }
      }

      container client-cert-auth {
        if-feature ssh-x509-certs;
        description
          "A reference to a list of trusted certificate authority (CA)
           certificates and a reference to a list of trusted client
           certificates.";
        leaf trusted-ca-certs {
          type leafref {
            path "/kc:keychain/kc:trusted-certificates/kc:name";
          }
          description
            "A reference to a list of certificate authority (CA)
             certificates used by the SSH server to authenticate
             SSH client certificates.";
        }

        leaf trusted-client-certs {
          type leafref {
            path "/kc:keychain/kc:trusted-certificates/kc:name";
          }
          description
            "A reference to a list of client certificates used by
             the SSH server to authenticate SSH client certificates.
             A clients certificate is authenticated if it is an
             exact match to a configured trusted client certificate.";
        }
      }
    }


    grouping listening-ssh-server-grouping {
      description
        "A reusable grouping for a SSH server that can be used as a
         basis for specific SSH server instances.";
      leaf address {
        type inet:ip-address;
        description
         "The IP address of the interface to listen on. The SSH
          server will listen on all interfaces if no value is
          specified.";
      }
      leaf port {
        type inet:port-number;
        mandatory true;  // will a default augmented in work?
        description
         "The local port number on this interface the SSH server
```

```
          listens on.";
      }
      uses non-listening-ssh-server-grouping;
    }

    container listening-ssh-server {
      description
        "This container will be removed by the RFC Editor. This
         container is currently only present in order to enable
         the ‘pyang‘ tool to generate tree diagram output of this
         module (used in the draft) as it otherwise would not
         contain any protocol accessible nodes to output.";

         uses listening-ssh-server-grouping;
    }
  }



  <CODE ENDS>
```

4.3. The TLS Server Model

The TLS Server model presented in this section presents two YANG groupings, one for a server that opens a socket to accept TCP connections on, and another for a server that has had the TCP connection opened for it already (e.g., inetd).

The TLS Server model (like the SSH Server model presented above) is provided as a grouping so that it can be used in different contexts. For instance, the NETCONF Server model presented in Section 4.4 uses one grouping to configure a NETCONF server listening for connections and the other grouping to configure NETCONF call home.

A shared characteristic between both groupings is the ability to configure which server certificate is presented to clients, the private key for which is held in the keychain model presented in Section 4.1. Another shared characteristic is the ability to configure which trusted CA or client certificates the server should be used to authenticate clients.

4.3.1. Tree Diagram

The following tree diagram represents the data model for the grouping used to configure an TLS server to listen for TCP connections. The tree diagram for the other grouping is not provided, but it is the same except without the "address" and "port" fields.

NOTE: the diagram below shows "listening-ssh-server" as a YANG container (not a grouping). This temporary container was created only to enable the ‘pyang‘ tool to output the tree diagram, as groupings by themselves have no protocol accessible nodes, and hence ‘pyang‘ would output an empty tree diagram.

```
 module: ietf-tls-server
    +--rw listening-tls-server
       +--rw address?        inet:ip-address
       +--rw port            inet:port-number
       +--rw certificates
       |  +--rw certificate* [name]
       |     +--rw name    -> /kc:keychain/private-keys/private-key/cert
 ificate-chains/certificate-chain/certificate
       +--rw client-auth
          +--rw trusted-ca-certs?       -> /kc:keychain/trusted-certific
 ates/name
          +--rw trusted-client-certs?   -> /kc:keychain/trusted-certific
 ates/name
```

4.3.2. Example Usage

```
   <listening-tls-server
     xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
     <port>6513</port>
     <certificates>
       <certificate>
         <name>ex-key-sect571r1-cert</name>
       </certificate>
     </certificates>
     <client-auth>
       <trusted-ca-certs>
         deployment-specific-ca-certs
       </trusted-ca-certs>
       <trusted-client-certs>
         explicitly-trusted-client-certs
       </trusted-client-certs>
     </client-auth>
   </listening-tls-server>
```

4.3.3. YANG Model

```
   <CODE BEGINS> file "ietf-tls-server@2016-03-16.yang"

   module ietf-tls-server {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
     prefix "ts";

     import ietf-inet-types {           // RFC 6991
       prefix inet;
     }
     import ietf-system-keychain {
       prefix kc;                       // RFC VVVV
       revision-date 2016-03-16;
     }

     organization
      "IETF NETCONF (Network Configuration) Working Group";

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

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

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

       Editor:   Kent Watsen
                 <mailto:kwatsen@juniper.net>";


     description
      "This module defines a reusable grouping for a TLS server that
       can be used as a basis for specific TLS server instances.

       Copyright (c) 2014 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
       (http://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC VVVV; see
       the RFC itself for full legal notices.";

     revision "2016-03-16" {
       description
        "Initial version";
       reference
        "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                   Models";
     }


     // grouping
     grouping non-listening-tls-server-grouping {
       description
         "A reusable grouping for a TLS server that can be used as a
          basis for specific TLS server instances.";
       container certificates {
         description
           "The list of certificates the TLS server will present when
            establishing a TLS connection in its Certificate message,
            as defined in Section 7.4.2 in RRC 5246.";
         reference
           "RFC 5246:
              The Transport Layer Security (TLS) Protocol Version 1.2";
         list certificate {
           key name;
           min-elements 1;
           description
             "An unordered list of certificates the TLS server can pick
              from when sending its Server Certificate message.";
           reference
             "RFC 5246: The TLS Protocol, Section 7.4.2";
           leaf name {
             type leafref {
               path "/kc:keychain/kc:private-keys/kc:private-key/"
                    + "kc:certificate-chains/kc:certificate-chain/"
                    + "kc:certificate";
             }
             description
               "The name of the certificate in the keychain.";
           }
         }
       }

       container client-auth {
         description
           "A reference to a list of trusted certificate authority (CA)
            certificates and a reference to a list of trusted client
            certificates.";
         leaf trusted-ca-certs {
           type leafref {
             path "/kc:keychain/kc:trusted-certificates/kc:name";
           }
           description
             "A reference to a list of certificate authority (CA)
              certificates used by the TLS server to authenticate
              TLS client certificates.";
         }

         leaf trusted-client-certs {
           type leafref {
             path "/kc:keychain/kc:trusted-certificates/kc:name";
           }
           description
             "A reference to a list of client certificates used by
              the TLS server to authenticate TLS client certificates.
              A clients certificate is authenticated if it is an
              exact match to a configured trusted client certificate.";
         }
       }
     }


     grouping listening-tls-server-grouping {
       description
         "A reusable grouping for a TLS server that can be used as a
          basis for specific TLS server instances.";
       leaf address {
         type inet:ip-address;
         description
          "The IP address of the interface to listen on. The TLS
           server will listen on all interfaces if no value is
           specified.";
       }
       leaf port {
         type inet:port-number;
         mandatory true;  // will a default augmented in work?
         description
          "The local port number on this interface the TLTLS server
           listens on.";
       }
       uses non-listening-tls-server-grouping;
     }

     container listening-tls-server {
       description
         "This container will be removed by the RFC Editor. This
          container is currently only present in order to enable
          the ‘pyang‘ tool to generate tree diagram output of this
          module (used in the draft) as it otherwise would not
          contain any protocol accessible nodes to output.";
```

```
          uses listening-tls-server-grouping;
     }
   }



   <CODE ENDS>
```

4.4. The NETCONF Server Model

The NETCONF Server model presented in this section supports servers both listening for connections to accept as well as initiating call-home connections. This model also supports both the SSH and TLS transport protocols, using the SSH Server and TLS Server groupings presented in Section 4.2 and Section 4.3 respectively. All private keys and trusted certificates are held in the keychain model presented in Section 4.1. YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF server supports.

4.4.1. Tree Diagram

The following tree diagram uses line-wrapping in order to comply with xml2rfc validation. This is annoying as I find that drafts (even txt drafts) look just fine with long lines - maybe xml2rfc should remove this warning? - or pyang could have an option to suppress printing leafref paths?

```
 module: ietf-netconf-server
    +--rw netconf-server
       +--rw session-options
       |  +--rw hello-timeout?   uint16
       +--rw listen {(ssh-listen or tls-listen)}?
       |  +--rw max-sessions?   uint16
       |  +--rw idle-timeout?   uint16
       |  +--rw endpoint* [name]
       |     +--rw name    string
       |     +--rw (transport)
       |        +--:(ssh) {ssh-listen}?
       |        |  +--rw ssh
       |        |     +--rw address?            inet:ip-address
       |        |     +--rw port                inet:port-number
       |        |     +--rw host-keys
       |        |     |  +--rw host-key* [name]
       |        |     |     +--rw name            string
       |        |     |     +--rw (type)?
       |        |     |        +--:(public-key)
       |        |     |        |  +--rw public-key?    -> /kc:keychain/p
 rivate-keys/private-key/name
       |        |     |        +--:(certificate)
       |        |     |           +--rw certificate?   -> /kc:keychain/p
 rivate-keys/private-key/certificate-chains/certificate-chain/certificat
 e {ssh-x509-certs}?
       |        |     +--rw client-cert-auth {ssh-x509-certs}?
       |        |        +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
       |        |        +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
       |        +--:(tls) {tls-listen}?
       |           +--rw tls
       |              +--rw address?        inet:ip-address
       |              +--rw port            inet:port-number
       |              +--rw certificates
       |              |  +--rw certificate* [name]
       |              |     +--rw name    -> /kc:keychain/private-keys/p
 rivate-key/certificate-chains/certificate-chain/certificate
       |              +--rw client-auth
       |                 +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
       |                 +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
       |                 +--rw cert-maps
       |                    +--rw cert-to-name* [id]
       |                       +--rw id             uint32
       |                       +--rw fingerprint    x509c2n:tls-fingerpr
 int
       |                       +--rw map-type       identityref
       |                       +--rw name           string
       +--rw call-home {(ssh-call-home or tls-call-home)}?
          +--rw netconf-client* [name]
             +--rw name                  string
             +--rw (transport)
             |  +--:(ssh) {ssh-call-home}?
             |  |  +--rw ssh
             |  |     +--rw endpoints
             |  |     |  +--rw endpoint* [name]
             |  |     |     +--rw name       string
             |  |     |     +--rw address    inet:host
             |  |     |     +--rw port?      inet:port-number
             |  |     +--rw host-keys
             |  |     |  +--rw host-key* [name]
             |  |     |     +--rw name           string
             |  |     |     +--rw (type)?
             |  |     |        +--:(public-key)
             |  |     |        |  +--rw public-key?    -> /kc:keychain/p
 rivate-keys/private-key/name
```

```
             |  |     |        +--:(certificate)
             |  |     |           +--rw certificate?   -> /kc:keychain/p
 rivate-keys/private-key/certificate-chains/certificate-chain/certificat
 e {ssh-x509-certs}?
             |  |     +--rw client-cert-auth {ssh-x509-certs}?
             |  |        +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
             |  |        +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
             |  +--:(tls) {tls-call-home}?
             |     +--rw tls
             |        +--rw endpoints
             |        |  +--rw endpoint* [name]
             |        |     +--rw name       string
             |        |     +--rw address    inet:host
             |        |     +--rw port?      inet:port-number
             |        +--rw certificates
             |        |  +--rw certificate* [name]
             |        |     +--rw name    -> /kc:keychain/private-keys/p
 rivate-key/certificate-chains/certificate-chain/certificate
             |        +--rw client-auth
             |           +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw cert-maps
             |              +--rw cert-to-name* [id]
             |                 +--rw id             uint32
             |                 +--rw fingerprint    x509c2n:tls-fingerpr
 int
             |                 +--rw map-type       identityref
             |                 +--rw name           string
             +--rw connection-type
             |  +--rw (connection-type)?
             |     +--:(persistent-connection)
             |     |  +--rw persistent!
             |     |     +--rw idle-timeout?   uint32
             |     |     +--rw keep-alives
             |     |        +--rw max-wait?       uint16
             |     |        +--rw max-attempts?   uint8
             |     +--:(periodic-connection)
             |        +--rw periodic!
             |           +--rw idle-timeout?        uint16
             |           +--rw reconnect_timeout?   uint16
             +--rw reconnect-strategy
                +--rw start-with?     enumeration
                +--rw max-attempts?   uint8
```

4.4.2. Example Usage

Configuring a NETCONF Server to listen for NETCONF client connections using both the SSH and TLS transport protocols, as well as configuring call-home to two NETCONF clients, one using SSH and the other using TLS.

This example is consistent with other examples presented in this document.

```
<netconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <listen>

    <!-- listening for SSH connections -->
    <endpoint>
      <name>netconf/ssh</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>
            <public-key>my-rsa-key</public-key>
          </host-key>
          <host-key>
            <certificate>TPM key</certificate>
          </host-key>
        </host-keys>
        <client-cert-auth>
          <trusted-ca-certs>
            deployment-specific-ca-certs
          </trusted-ca-certs>
          <trusted-client-certs>
            explicitly-trusted-client-certs
          </trusted-client-certs>
        </client-cert-auth>
      </ssh>
    </endpoint>

    <!-- listening for TLS connections -->
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>ex-key-sect571r1-cert</certificate>
        </certificates>
        <client-auth>
          <trusted-ca-certs>
```

```
               deployment-specific-ca-certs
             </trusted-ca-certs>
             <trusted-client-certs>
               explicitly-trusted-client-certs
             </trusted-client-certs>
             <cert-maps>
               <cert-to-name>
                 <id>1</id>
                 <fingerprint>11:0A:05:11:00</fingerprint>
                 <map-type>x509c2n:san-any</map-type>
               </cert-to-name>
               <cert-to-name>
                 <id>2</id>
                 <fingerprint>B3:4F:A1:8C:54</fingerprint>
                 <map-type>x509c2n:specified</map-type>
                 <name>scooby-doo</name>
               </cert-to-name>
             </cert-maps>
           </client-auth>
         </tls>
       </endpoint>

     </listen>
     <call-home>

       <!-- calling home to an SSH-based NETCONF client -->
       <netconf-client>
         <name>config-mgr</name>
         <ssh>
           <endpoints>
             <endpoint>
               <name>east-data-center</name>
               <address>11.22.33.44</address>
             </endpoint>
             <endpoint>
               <name>west-data-center</name>
               <address>55.66.77.88</address>
             </endpoint>
           </endpoints>
           <host-keys>
             <host-key>
               <certificate>TPM key</certificate>
             </host-key>
           </host-keys>
           <client-cert-auth>
             <trusted-ca-certs>
               deployment-specific-ca-certs
             </trusted-ca-certs>
             <trusted-client-certs>
               explicitly-trusted-client-certs
             </trusted-client-certs>
           </client-cert-auth>
         </ssh>
         <connection-type>
           <periodic>
             <idle-timeout>300</idle-timeout>
             <reconnect-timeout>60</reconnect-timeout>
           </periodic>
         </connection-type>
         <reconnect-strategy>
           <start-with>last-connected</start-with>
           <max-attempts>3</max-attempts>
         </reconnect-strategy>
       </netconf-client>

       <!-- calling home to a TLS-based NETCONF client -->
       <netconf-client>
         <name>event-correlator</name>
         <tls>
           <endpoints>
             <endpoint>
               <name>east-data-center</name>
               <address>22.33.44.55</address>
             </endpoint>
             <endpoint>
               <name>west-data-center</name>
               <address>33.44.55.66</address>
             </endpoint>
           </endpoints>
           <certificates>
             <certificate>ex-key-sect571r1-cert</certificate>
           </certificates>
           <client-auth>
             <trusted-ca-certs>
               deployment-specific-ca-certs
             </trusted-ca-certs>
             <trusted-client-certs>
               explicitly-trusted-client-certs
             </trusted-client-certs>
             <cert-maps>
               <cert-to-name>
                 <id>1</id>
                 <fingerprint>11:0A:05:11:00</fingerprint>
                 <map-type>x509c2n:san-any</map-type>
               </cert-to-name>
               <cert-to-name>
```

```
                 <id>2</id>
                 <fingerprint>B3:4F:A1:8C:54</fingerprint>
                 <map-type>x509c2n:specified</map-type>
                 <name>scooby-doo</name>
               </cert-to-name>
             </cert-maps>
           </client-auth>
         </tls>
         <connection-type>
           <persistent>
             <idle-timeout>300</idle-timeout>
             <keep-alives>
               <max-wait>30</max-wait>
               <max-attempts>3</max-attempts>
             </keep-alives>
           </persistent>
         </connection-type>
         <reconnect-strategy>
           <start-with>first-listed</start-with>
           <max-attempts>3</max-attempts>
         </reconnect-strategy>
       </netconf-client>

     </call-home>
   </netconf-server>
```

4.4.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

```
  <CODE BEGINS> file "ietf-netconf-server@2016-03-16.yang"

  module ietf-netconf-server {
    yang-version 1.1;

    namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-server";
    prefix "ncserver";

    import ietf-inet-types {           // RFC 6991
      prefix inet;
    }
    import ietf-x509-cert-to-name {    // RFC 7407
      prefix x509c2n;
    }
    import ietf-ssh-server {           // RFC VVVV
      prefix ss;
      revision-date 2016-03-16;
    }
    import ietf-tls-server {           // RFC VVVV
      prefix ts;
      revision-date 2016-03-16;
    }


    organization
     "IETF NETCONF (Network Configuration) Working Group";

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

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

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

      Editor:   Kent Watsen
                <mailto:kwatsen@juniper.net>";


    description
     "This module contains a collection of YANG definitions for
      configuring NETCONF servers.

      Copyright (c) 2014 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
      (http://trustee.ietf.org/license-info).

      This version of this YANG module is part of RFC VVVV; see
      the RFC itself for full legal notices.";

    revision "2016-03-16" {
      description
       "Initial version";
      reference
       "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                  Models";
    }
    // Features

    feature ssh-listen {
      description
       "The ssh-listen feature indicates that the NETCONF server
        supports opening a port to accept NETCONF over SSH
        client connections.";
      reference
       "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
    }

    feature ssh-call-home {
      description
       "The ssh-call-home feature indicates that the NETCONF
        server supports initiating a NETCONF over SSH call
        home connection to NETCONF clients.";
      reference
       "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
    }

    feature tls-listen {
      description
       "The tls-listen feature indicates that the NETCONF server
        supports opening a port to accept NETCONF over TLS
        client connections.";
      reference
       "RFC 7589: Using the NETCONF Protocol over Transport
                  Layer Security (TLS) with Mutual X.509
                  Authentication";
    }

    feature tls-call-home {
      description
       "The tls-call-home feature indicates that the NETCONF
        server supports initiating a NETCONF over TLS call
        home connection to NETCONF clients.";
      reference
       "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
    }

    feature ssh-x509-certs {
      description
        "The ssh-x509-certs feature indicates that the NETCONF
         server supports RFC 6187";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
         Authentication";
    }
    // top-level container  (groupings below)
    container netconf-server {
      description
        "Top-level container for NETCONF server configuration.";

      container session-options {  // SHOULD WE REMOVE THIS ALTOGETHER?
        description
          "NETCONF session options, independent of transport
           or connection strategy.";
        leaf hello-timeout {
          type uint16;
          units "seconds";
          default 600;
          description
            "Specifies the maximum number of seconds that a SSH/TLS
             connection may wait for a hello message to be received.
             A connection will be dropped if no hello message is
             received before this number of seconds elapses.  If set
             to zero, then the server will wait forever for a hello
             message.";
        }
      }

      container listen {
        if-feature "(ssh-listen or tls-listen)";
        description
          "Configures listen behavior";
        leaf max-sessions {
          type uint16;
          default 0;
          description
            "Specifies the maximum number of concurrent sessions
             that can be active at one time.  The value 0 indicates
             that no artificial session limit should be used.";
        }
        leaf idle-timeout {
          type uint16;
          units "seconds";
          default 3600; // one hour
          description
            "Specifies the maximum number of seconds that a NETCONF
             session may remain idle. A NETCONF session will be dropped
             if it is idle for an interval longer than this number of
             seconds.  If set to zero, then the server will never drop
             a session because it is idle.  Sessions that have a
             notification subscription active are never dropped.";
        }
        list endpoint {
          key name;
          description
            "List of endpoints to listen for NETCONF connections on.";
          leaf name {
            type string;
            description
              "An arbitrary name for the NETCONF listen endpoint.";
          }
          choice transport {
            mandatory true;
            description
              "Selects between available transports.";
            case ssh {
              if-feature ssh-listen;
              container ssh {
                description
                  "SSH-specific listening configuration for inbound
                   connections.";
                uses ss:listening-ssh-server-grouping {
                  refine port {
                    default 830;
                  }
                }
              }
            }
            case tls {
              if-feature tls-listen;
              container tls {
                description
                  "TLS-specific listening configuration for inbound
                   connections.";
                uses ts:listening-tls-server-grouping {
                  refine port {
                    default 6513;
                  }
                  augment "client-auth" {
                    description
                      "Augments in the cert-to-name structure.";
                    uses cert-maps-grouping;
                  }
                }
              }
            }
          }
        }
      }

      container call-home {
        if-feature "(ssh-call-home or tls-call-home)";
        description
          "Configures call-home behavior";
        list netconf-client {
          key name;
          description
            "List of NETCONF clients the NETCONF server is to initiate
             call-home connections to.";
          leaf name {
            type string;
            description
              "An arbitrary name for the remote NETCONF client.";
          }
          choice transport {
            mandatory true;
            description
              "Selects between available transports.";
            case ssh {
              if-feature ssh-call-home;
              container ssh {
                description
                  "Specifies SSH-specific call-home transport
                   configuration.";
                uses endpoints-container {
                  refine endpoints/endpoint/port {
                    default 7777;
                  }
                }
                uses ss:non-listening-ssh-server-grouping;
              }
            }
            case tls {
              if-feature tls-call-home;
              container tls {
                description
                  "Specifies TLS-specific call-home transport
                   configuration.";
                uses endpoints-container {
                  refine endpoints/endpoint/port {
                    default 8888;
                  }
                }
                uses ts:non-listening-tls-server-grouping {
                  augment "client-auth" {
                    description
                      "Augments in the cert-to-name structure.";
                    uses cert-maps-grouping;
                  }
                }
              }
            }
          }
          container connection-type {
            description
             "Indicates the kind of connection to use.";
            choice connection-type {
              description
                "Selects between available connection types.";
              case persistent-connection {
                container persistent {
                  presence true;
                  description
                   "Maintain a persistent connection to the NETCONF
                    client. If the connection goes down, immediately
                    start trying to reconnect to it, using the
                    reconnection strategy.

                    This connection type minimizes any NETCONF client
                    to NETCONF server data-transfer delay, albeit at
                    the expense of holding resources longer.";
                  leaf idle-timeout {
                    type uint32;
                    units "seconds";
                    default 86400;  // one day;
                    description
                      "Specifies the maximum number of seconds that a
                       a NETCONF session may remain idle. A NETCONF
                       session will be dropped if it is idle for an
                       interval longer than this number of seconds.
                       If set to zero, then the server will never drop
                       a session because it is idle.  Sessions that
                       have a notification subscription active are
                       never dropped.";
                  }
                  container keep-alives {
                    description
                      "Configures the keep-alive policy, to proactively
                       test the aliveness of the SSH/TLS client.  An
                       unresponsive SSH/TLS client will be dropped after
                       approximately max-attempts * max-wait seconds.";
                    reference
                      "RFC YYYY: NETCONF Call Home and RESTCONF Call
                       Home, Section 3.1, item S6";
                    leaf max-wait {
                      type uint16 {
                        range "1..max";
                      }
                      units seconds;
                      default 30;
                      description
                       "Sets the amount of time in seconds after which
                        if no data has been received from the SSH/TLS
                        client, a SSH/TLS-level message will be sent
                        to test the aliveness of the SSH/TLS client.";
                    }
                    leaf max-attempts {
                      type uint8;
                      default 3;
                      description
                       "Sets the number of maximum number of sequential
                        keep-alive messages that can fail to obtain a
                        response from the SSH/TLS client before assuming
                        the SSH/TLS client is no longer alive.";
                    }
                  }
                }
              }
              case periodic-connection {
                container periodic {
                  presence true;
                  description
                   "Periodically connect to the NETCONF client, so that
                    the NETCONF client may deliver messages pending for
                    the NETCONF server.  The NETCONF client is expected
                    to close the connection when it is ready to release
                    it, thus starting the NETCONF server’s timer until
                    next connection.";
                  leaf idle-timeout {
                    type uint16;
                    units "seconds";
                    default 300; // five minutes
                    description
                      "Specifies the maximum number of seconds that a
                       a NETCONF session may remain idle. A NETCONF
                       session will be dropped if it is idle for an
                       interval longer than this number of seconds.
                       If set to zero, then the server will never drop
                       a session because it is idle.  Sessions that
                       have a notification subscription active are
                       never dropped.";
                  }
                  leaf reconnect_timeout {
                    type uint16 {
                      range "1..max";
                    }
                    units minutes;
                    default 60;
                    description
                     "Sets the maximum amount of unconnected time the
                      NETCONF server will wait before re-establishing
                      a connection to the NETCONF client.  The NETCONF
                      server may initiate a connection before this
                      time if desired (e.g., to deliver an event
                      notification message).";
                  }
                }
              }
            }
          }
          container reconnect-strategy {
            description
             "The reconnection strategy guides how a NETCONF server
              reconnects to a NETCONF client, after discovering its
              connection to the client has dropped.  The NETCONF
              server starts with the specified endpoint and tries
              to connect to it max-attempts times before trying the
              next endpoint in the list (round robin).";
            leaf start-with {
              type enumeration {
                enum first-listed {
                  description
                    "Indicates that reconnections should start with
                     the first endpoint listed.";
                }
                enum last-connected {
                  description
                    "Indicates that reconnections should start with
                     the endpoint last connected to.  If no previous
                     connection has ever been established, then the
                     first endpoint configured is used.   NETCONF
                     servers SHOULD be able to remember the last
                     endpoint connected to across reboots.";
                }
              }
              default first-listed;
              description
               "Specifies which of the NETCONF client’s endpoints the
                NETCONF server should start with when trying to connect
                to the NETCONF client.";
            }
            leaf max-attempts {
              type uint8 {
                range "1..max";
              }
              default 3;
              description
               "Specifies the number times the NETCONF server tries to
                connect to a specific endpoint before moving on to the
                next endpoint in the list (round robin).";
            }
          }
        }
      }
    }


    grouping cert-maps-grouping {
      description
        "A grouping that defines a container around the
         cert-to-name structure defined in RFC 7407.";
      container cert-maps {
        uses x509c2n:cert-to-name;
        description
         "The cert-maps container is used by a TLS-based NETCONF
          server to map the NETCONF client's presented X.509
          certificate to a NETCONF username.  If no matching and
          valid cert-to-name list entry can be found, then the
          NETCONF server MUST close the connection, and MUST NOT
          accept NETCONF messages over it.";
        reference
          "RFC WWWW: NETCONF over TLS, Section 7";
      }
    }


    grouping endpoints-container {
      description
        "This grouping is used by both the ssh and tls containers
         for call-home configurations.";
      container endpoints {
        description
          "Container for the list of endpoints.";
        list endpoint {
          key name;
          min-elements 1;
          ordered-by user;
          description
            "User-ordered list of endpoints for this NETCONF client.
             Defining more than one enables high-availability.";
          leaf name {
            type string;
            description
              "An arbitrary name for this endpoint.";
          }
          leaf address {
            type inet:host;
            mandatory true;
            description
             "The IP address or hostname of the endpoint.  If a
              hostname is configured and the DNS resolution results
              in more than one IP address, the NETCONF server
              will process the IP addresses as if they had been
              explicitly configured in place of the hostname.";
          }
          leaf port {
            type inet:port-number;
            description
             "The IP port for this endpoint. The NETCONF server will
              use the IANA-assigned well-known port if no value is
              specified.";
          }
        }
      }
    }

  }


  <CODE ENDS>
```

4.5. The RESTCONF Server Model

The RESTCONF Server model presented in this section supports servers both listening for connections to accept as well as initiating call-home connections. This model supports the TLS transport only, as RESTCONF only supports HTTPS, using the TLS Server groupings presented in Section 4.3. All private keys and trusted certificates are held in the keychain model presented in Section 4.1. YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF server supports.

4.5.1. Tree Diagram

The following tree diagram uses line-wrapping in order to comply with xml2rfc validation. This is annoying as I find that drafts (even txt drafts) look just fine with long lines - maybe xml2rfc should remove this warning? - or pyang could have an option to suppress printing leafref paths?

```
 module: ietf-restconf-server
    +--rw restconf-server
       +--rw listen {tls-listen}?
       |  +--rw max-sessions?   uint16
       |  +--rw endpoint* [name]
       |     +--rw name    string
       |     +--rw (transport)
       |        +--:(tls) {tls-listen}?
       |           +--rw tls
       |              +--rw address?        inet:ip-address
       |              +--rw port            inet:port-number
       |              +--rw certificates
       |              |  +--rw certificate* [name]
       |              |     +--rw name    -> /kc:keychain/private-keys/p
 rivate-key/certificate-chains/certificate-chain/certificate
       |              +--rw client-auth
       |                 +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
       |                 +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
       |                 +--rw cert-maps
       |                    +--rw cert-to-name* [id]
       |                       +--rw id             uint32
       |                       +--rw fingerprint    x509c2n:tls-fingerpr
 int
       |                       +--rw map-type       identityref
       |                       +--rw name           string
       +--rw call-home {tls-call-home}?
          +--rw restconf-client* [name]
             +--rw name                  string
             +--rw (transport)
             |  +--:(tls) {tls-call-home}?
             |     +--rw tls
             |        +--rw endpoints
             |        |  +--rw endpoint* [name]
             |        |     +--rw name       string
             |        |     +--rw address    inet:host
             |        |     +--rw port?      inet:port-number
             |        +--rw certificates
             |        |  +--rw certificate* [name]
             |        |     +--rw name    -> /kc:keychain/private-keys/p
 rivate-key/certificate-chains/certificate-chain/certificate
             |        +--rw client-auth
             |           +--rw trusted-ca-certs?       -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw trusted-client-certs?   -> /kc:keychain/t
 rusted-certificates/name
             |           +--rw cert-maps
             |              +--rw cert-to-name* [id]
             |                 +--rw id                uint32
             |                 +--rw fingerprint       x509c2n:tls-fingerpr
 int
             |                 +--rw map-type          identityref
             |                 +--rw name              string
             +--rw connection-type
             |  +--rw (connection-type)?
             |     +--:(persistent-connection)
             |     |  +--rw persistent!
             |     |     +--rw keep-alives
             |     |        +--rw max-wait?        uint16
             |     |        +--rw max-attempts?    uint8
             |     +--:(periodic-connection)
             |        +--rw periodic!
             |           +--rw reconnect-timeout?   uint16
             +--rw reconnect-strategy
                +--rw start-with?     enumeration
                +--rw max-attempts?   uint8
```

4.5.2. Example Usage

Configuring a RESTCONF Server to listen for RESTCONF client connections, as well as configuring call-home to one RESTCONF client.

This example is consistent with other examples presented in this document.

```
<restconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server">

  <!-- listening for TLS (HTTPS) connections -->
  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>ex-key-sect571r1-cert</certificate>
        </certificates>
        <client-auth>
          <trusted-ca-certs>
            deployment-specific-ca-certs
          </trusted-ca-certs>
          <trusted-client-certs>
            explicitly-trusted-client-certs
          </trusted-client-certs>
          <cert-maps>
```

```
               <cert-to-name>
                 <id>1</id>
                 <fingerprint>11:0A:05:11:00</fingerprint>
                 <map-type>x509c2n:san-any</map-type>
               </cert-to-name>
               <cert-to-name>
                 <id>2</id>
                 <fingerprint>B3:4F:A1:8C:54</fingerprint>
                 <map-type>x509c2n:specified</map-type>
                 <name>scooby-doo</name>
               </cert-to-name>
             </cert-maps>
           </client-auth>
         </tls>

       </endpoint>
     </listen>

     <!-- calling home to a RESTCONF client -->
     <call-home>
       <restconf-client>
         <name>config-manager</name>
         <tls>
           <endpoints>
             <endpoint>
               <name>east-data-center</name>
               <address>22.33.44.55</address>
             </endpoint>
             <endpoint>
               <name>west-data-center</name>
               <address>33.44.55.66</address>
             </endpoint>
           </endpoints>
           <certificates>
             <certificate>ex-key-sect571r1-cert</certificate>
           </certificates>
           <client-auth>
             <trusted-ca-certs>
               deployment-specific-ca-certs
             </trusted-ca-certs>
             <trusted-client-certs>
               explicitly-trusted-client-certs
             </trusted-client-certs>
             <cert-maps>
               <cert-to-name>
                 <id>1</id>
                 <fingerprint>11:0A:05:11:00</fingerprint>
                 <map-type>x509c2n:san-any</map-type>
               </cert-to-name>
               <cert-to-name>
                 <id>2</id>
                 <fingerprint>B3:4F:A1:8C:54</fingerprint>
                 <map-type>x509c2n:specified</map-type>
                 <name>scooby-doo</name>
               </cert-to-name>
             </cert-maps>
           </client-auth>
         </tls>
         <connection-type>
           <periodic>
             <idle-timeout>300</idle-timeout>
             <reconnect-timeout>60</reconnect-timeout>
           </periodic>
         </connection-type>
         <reconnect-strategy>
           <start-with>last-connected</start-with>
           <max-attempts>3</max-attempts>
         </reconnect-strategy>
       </restconf-client>
     </call-home>

   </restconf-server>
```

4.5.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

```
<CODE BEGINS> file "ietf-restconf-server@2016-03-16.yang"

module ietf-restconf-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-server";
  prefix "rcserver";

  //import ietf-netconf-acm {
  //  prefix nacm;                     // RFC 6536
  //}
  import ietf-inet-types {           // RFC 6991
    prefix inet;
  }
  import ietf-x509-cert-to-name {    // RFC 7407
    prefix x509c2n;
  }
  import ietf-tls-server {           // RFC VVVV
```

```
    prefix ts;
    revision-date 2016-03-16;
  }

  organization
   "IETF NETCONF (Network Configuration) Working Group";

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

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

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

    Editor:   Kent Watsen
              <mailto:kwatsen@juniper.net>";


  description
   "This module contains a collection of YANG definitions for
    configuring RESTCONF servers.

    Copyright (c) 2014 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
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC VVVV; see
    the RFC itself for full legal notices.";

  revision "2016-03-16" {
    description
     "Initial version";
    reference
     "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                Models";
  }


  // Features
  feature tls-listen {
    description
     "The listen feature indicates that the RESTCONF server
      supports opening a port to listen for incoming RESTCONF
      client connections.";
    reference
     "RFC XXXX: RESTCONF Protocol";
  }

  feature tls-call-home {
    description
     "The call-home feature indicates that the RESTCONF server
      supports initiating connections to RESTCONF clients.";
    reference
     "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
  }

  feature client-cert-auth {
    description
     "The client-cert-auth feature indicates that the RESTCONF
      server supports the ClientCertificate authentication scheme.";
    reference
     "RFC ZZZZ: Client Authentication over New TLS Connection";
  }


  // top-level container
  container restconf-server {
    description
      "Top-level container for RESTCONF server configuration.";

    container listen {
      if-feature tls-listen;
      description
        "Configures listen behavior";
      leaf max-sessions {
        type uint16;
        default 0;   // should this be ’max’?
        description
          "Specifies the maximum number of concurrent sessions
           that can be active at one time.  The value 0 indicates
           that no artificial session limit should be used.";
      }
      list endpoint {
        key name;
        description
          "List of endpoints to listen for RESTCONF connections on.";
        leaf name {
          type string;
          description
            "An arbitrary name for the RESTCONF listen endpoint.";
        }
        choice transport {
          mandatory true;
          description
            "Selects between available transports.";
          case tls {
            if-feature tls-listen;
            container tls {
              description
                "TLS-specific listening configuration for inbound
                 connections.";
              uses ts:listening-tls-server-grouping {
                refine port {
                  default 443;
                }
                augment "client-auth" {
                  description
                    "Augments in the cert-to-name structure.";
                  uses cert-maps-grouping;
                }
              }
            }
          }
        }
      }
    }

    container call-home {
      if-feature tls-call-home;
      description
        "Configures call-home behavior";
      list restconf-client {
        key name;
        description
          "List of RESTCONF clients the RESTCONF server is to
           initiate call-home connections to.";
        leaf name {
          type string;
          description
            "An arbitrary name for the remote RESTCONF client.";
        }
        choice transport {
          mandatory true;
          description
            "Selects between TLS and any transports augmented in.";
          case tls {
            if-feature tls-call-home;
            container tls {
              description
                "Specifies TLS-specific call-home transport
                 configuration.";
              uses endpoints-container {
                refine endpoints/endpoint/port {
                  default 9999;
                }
              }
              uses ts:non-listening-tls-server-grouping {
                augment "client-auth" {
                  description
                    "Augments in the cert-to-name structure.";
                  uses cert-maps-grouping;
                }
              }
            }
          }
        }
        container connection-type {
          description
           "Indicates the RESTCONF client’s preference for how the
            RESTCONF server’s connection is maintained.";
          choice connection-type {
            description
              "Selects between available connection types.";
            case persistent-connection {
              container persistent {
                presence true;
                description
                 "Maintain a persistent connection to the RESTCONF
                  client. If the connection goes down, immediately
                  start trying to reconnect to it, using the
                  reconnection strategy.

                  This connection type minimizes any RESTCONF client
                  to RESTCONF server data-transfer delay, albeit at
                  the expense of holding resources longer.";

                container keep-alives {
                  description
                    "Configures the keep-alive policy, to proactively
                     test the aliveness of the TLS client.  An
                     unresponsive TLS client will be dropped after
                     approximately (max-attempts * max-wait) seconds.";
                  reference
```

```
                    "RFC YYYY: NETCONF Call Home and RESTCONF Call Home,
                     Section 3.1, item S6";
                  leaf max-wait {
                    type uint16 {
                      range "1..max";
                    }
                    units seconds;
                    default 30;
                    description
                     "Sets the amount of time in seconds after which
                      if no data has been received from the TLS
                      client, a TLS-level message will be sent to
                      test the aliveness of the TLS client.";
                  }
                  leaf max-attempts {
                    type uint8;
                    default 3;
                    description
                     "Sets the number of sequential keep-alive messages
                      that can fail to obtain a response from the TLS
                      client before assuming the TLS client is no
                      longer alive.";
                  }
                }
              }
            }
            case periodic-connection {
              container periodic {
                presence true;
                description
                 "Periodically connect to the RESTCONF client, so that
                  the RESTCONF client may deliver messages pending for
                  the RESTCONF server.  The RESTCONF client is expected
                  to close the connection when it is ready to release
                  it, thus starting the RESTCONF server’s timer until
                  next connection.";
                leaf reconnect-timeout {
                  type uint16 {
                    range "1..max";
                  }
                  units minutes;
                  default 60;
                  description
                   "The maximum amount of unconnected time the RESTCONF
                    server will wait before re-establishing a connection
                    to the RESTCONF client.  The RESTCONF server may
                    initiate a connection before this time if desired
                    (e.g., to deliver a notification).";
                }
              }
            }
          }
        }
        container reconnect-strategy {
          description
           "The reconnection strategy guides how a RESTCONF server
            reconnects to an RESTCONF client, after losing a connection
            to it, even if due to a reboot.  The RESTCONF server starts
            with the specified endpoint and tries to connect to it
            max-attempts times before trying the next endpoint in the
            list (round robin).";
          leaf start-with {
            type enumeration {
              enum first-listed {
                description
                  "Indicates that reconnections should start with
                   the first endpoint listed.";
              }
              enum last-connected {
                description
                  "Indicates that reconnections should start with
                   the endpoint last connected to.  If no previous
                   connection has ever been established, then the
                   first endpoint configured is used.   RESTCONF
                   servers SHOULD be able to remember the last
                   endpoint connected to across reboots.";
              }
            }
            default first-listed;
            description
             "Specifies which of the RESTCONF client's endpoints the
              RESTCONF server should start with when trying to connect
              to the RESTCONF client.";
          }
          leaf max-attempts {
            type uint8 {
              range "1..max";
            }
            default 3;
            description
             "Specifies the number times the RESTCONF server tries to
              connect to a specific endpoint before moving on to the
              next endpoint in the list (round robin).";
          }
        }
      }
```

```
    }
  }


  grouping cert-maps-grouping {
    description
      "A grouping that defines a container around the
       cert-to-name structure defined in RFC 7407.";
    container cert-maps {
      uses x509c2n:cert-to-name;
      description
       "The cert-maps container is used by a TLS-based RESTCONF
        server to map the RESTCONF client's presented X.509
        certificate to a RESTCONF username.  If no matching and
        valid cert-to-name list entry can be found, then the
        RESTCONF server MUST close the connection, and MUST NOT
        accept RESTCONF messages over it.";
      reference
        "RFC XXXX: The RESTCONF Protocol";
    }
  }


  grouping endpoints-container {
    description
      "This grouping is used by tls container for call-home
       configurations.";
    container endpoints {
      description
        "Container for the list of endpoints.";
      list endpoint {
        key name;
        min-elements 1;
        ordered-by user;
        description
          "User-ordered list of endpoints for this RESTCONF client.
           Defining more than one enables high-availability.";
        leaf name {
          type string;
          description
            "An arbitrary name for this endpoint.";
        }
        leaf address {
          type inet:host;
          mandatory true;
          description
           "The IP address or hostname of the endpoint.  If a
            hostname is configured and the DNS resolution results
            in more than one IP address, the RESTCONF server
            will process the IP addresses as if they had been
            explicitly configured in place of the hostname.";
        }
        leaf port {
          type inet:port-number;
          description
           "The IP port for this endpoint. The RESTCONF server will
            use the IANA-assigned well-known port if no value is
            specified.";
        }
      }
    }
  }


}

<CODE ENDS>
```

5. Design Considerations

The manner that the both local and remote endpoints have been specified in the ietf-netconf-server and ietf-rest-server modules does not directly support virtual routing and forwarding (VRF), though they have been specified in such a way to enable external modules will augment in VRF designations when needed.

This document uses PKCS #10 [RFC2986] for the "generate-certificate-signing-request" action. The use of Certificate Request Message Format (CRMF) [RFC4211] was considered, but is was unclear if there was market demand for it, and so support for CRMF has been left out of this specification. If it is desired to support CRMF in the future, placing a "choice" statement in both the input and output statements, along with an "if-feature" statement on the CRMF option, would enable a backwards compatible solution.

This document puts a limit of the number of elliptical curves supported. This was done to match industry trends in IETF best practice (e.g., matching work being done in TLS 1.3). In additional algorithms are needed, they MAY be augmented in by another module, or added directly in a future version of this document.

Both this document and Key Chain YANG Data Model [draft-ietf-rtgwg-yang-key-chain] define keychain YANG modules. The authors looked at this and agree that they two modules server different purposes and hence not worth merging into one document. To underscore this further, this document renamed its module from "ietf-keychain" to "ietf-system-keychain" and that other document renamed its module from "ietf-key-chain" to "ietf-routing-key-chain".

For the trusted-certificates list, Trust Anchor Format [RFC5914] was evaluated and deemed inappropriate due to this document’s need to also support pinning. That is, pinning a client-certificate to support NETCONF over TLS client authentication.

6. Security Considerations

This document defines a keychain mechanism that is entrusted with the safe keeping of private keys, and the safe keeping of trusted certificates. Nowhere in this API is there an ability to access (read out) a private key once it is known to the keychain. Further, associated public keys and attributes (e.g., algorithm name, key length, etc.) are read-only. That said, this document allows for the deletion of private keys and their certificates, as well the deletion of trusted certificates. Access control mechanisms (e.g., NACM [RFC6536]) MUST be in place so as to authorize such client actions. Further, whilst the data model allows for private keys and trusted certificates in general to be deleted, implementations should be well aware that some privates keys (e.g., those in a TPM) and some trusted certificates, should never be deleted, regardless if the authorization mechanisms would generally allow for such actions.

For the "generate-certificate-signing-request" action, it is RECOMMENDED that devices implement assert channel binding [RFC5056], so as to ensure that the application layer that sent the request is the same as the device authenticated in the secure transport layer was established.

This document defines a data model that includes a list of private keys. These private keys MAY be deleted using standard NETCONF or RESTCONF operations (e.g., <edit-config>). Implementations SHOULD automatically (without explicit request) zeroize these keys in the most secure manner available, so as to prevent the remnants of their persisted storage locations from being analyzed in any meaningful way.

The keychain module define within this document defines the "load-private-key" action enabling a device to load a client-supplied private key. This is a private key with no shrouding to protect it. The strength of this private key MUST NOT be greater than the strength of the underlying secure transport connection over which it is communicated. Devices SHOULD fail this request if ever the strength of the private key is greater then the strength of the underlying transport.

A denial of service (DoS) attack MAY occur if the NETCONF server limits the maximum number of NETCONF sessions it will accept (i.e. the ’max-sessions’ field in the ietf-netconf-server module is not zero) and either the "hello-timeout" or "idle-timeout" fields in ietf-netconf-server module have been set to indicate the NETCONF server should wait forever (i.e. set to zero).

7. IANA Considerations

7.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registrations are requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-netconf-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

7.2. The YANG Module Names Registry

This document registers five YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the the following registrations are requested:

```
      name:         ietf-system-keychain
      namespace:    urn:ietf:params:xml:ns:yang:ietf-system-keychain
      prefix:       kc
      reference:    RFC VVVV

      name:         ietf-ssh-server
      namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-server
      prefix:       ssvr
      reference:    RFC VVVV

      name:         ietf-tls-server
      namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-server
      prefix:       tsvr
      reference:    RFC VVVV

      name:         ietf-netconf-server
      namespace:    urn:ietf:params:xml:ns:yang:ietf-netconf-server
      prefix:       ncsvr
      reference:    RFC VVVV

      name:         ietf-restconf-server
      namespace:    urn:ietf:params:xml:ns:yang:ietf-restconf-server
      prefix:       rcsvr
      reference:    RFC VVVV
```

8. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Phil Shafer, Sean Turner, and Bert Wijnen.

Juergen Schoenwaelder and was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

9. References

9.1. Normative References

[draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ieft-netconf-call-home-02 (work in progress), 2014.

[draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ieft-netconf-restconf-04 (work in progress), 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <http://www.rfc-editor.org/info/rfc2986>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <http://www.rfc-editor.org/info/rfc4253>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <http://www.rfc-editor.org/info/rfc5958>.

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

[RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, DOI 10.17487/RFC6187, March 2011, <http://www.rfc-editor.org/info/rfc6187>.

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

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

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

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <http://www.rfc-editor.org/info/rfc7407>.

[RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <http://www.rfc-editor.org/info/rfc7589>.

9.2. Informative References

[draft-ietf-rtgwg-yang-key-chain] Lindem, A., Qu, Y., Yeung, D., Chen, I., Zhang, J., and Y. Yang, "Key Chain YANG Data Model", draft-ietf-rtgwg-yang-key-chain (work in progress), 2016, <https://tools.ietf.org/html/draft-ietf-rtgwg-yang-key-chain>.

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

[RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <http://www.rfc-editor.org/info/rfc4211>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <http://www.rfc-editor.org/info/rfc5056>.

[RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <http://www.rfc-editor.org/info/rfc5914>.

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

Appendix A. Change Log

A.1. 00 to 01

- Restructured document so it flows better

- Added trusted-ca-certs and trusted-client-certs objects into the ietf-system-tls-auth module

A.2. 01 to 02

- removed the "one-to-many" construct

- removed "address" as a key field

- removed "network-manager" terminology

- moved open issues to github issues

- brought TLS client auth back into model

A.3. 02 to 03

- fixed tree diagrams and surrounding text

A.4. 03 to 04

- reduced the number of grouping statements

- removed psk-maps and associated feature statements

- added ability for listen/call-home instances to specify which host-keys/certificates (of all listed) to use

- clarified that last-connected should span reboots

- added missing "objectives" for selecting which keys to use, authenticating client-certificates, and mapping authenticated client-certificates to usernames

- clarified indirect client certificate authentication

- added keep-alive configuration for listen connections

- added global-level NETCONF session parameters

A.5. 04 to 05

- Removed all refs to the old ietf-system-tls-auth module

- Removed YANG 1.1 style if-feature statements (loss some expressiveness)

- Removed the read-only (config false) lists of SSH host-keys and TLS certs

- Added an if-feature around session-options container

- Added ability to configure trust-anchors for SSH X.509 client certs

- Now imports by revision, per best practice

- Added support for RESTCONF server

- Added RFC Editor instructions

A.6. 05 to 06

- Removed feature statement on the session-options container (issue #21).

- Added NACM statements to YANG modules for sensitive nodes (issue #24).

- Fixed default RESTCONF server port value to be 443 (issue #26).

- Added client-cert-auth subtree to ietf-restconf-server module (issue #27).

- Updated draft-ietf-netmod-snmp-cfg reference to RFC 7407 (issue #28).

- Added description statements for groupings (issue #29).

- Added description for braces to tree diagram section (issue #30).

- Renamed feature from "rfc6187" to "ssh-x509-certs" (issue #31).

A.7. 06 to 07

- Replaced "application" with "NETCONF/RESTCONF client" (issue #32).

- Reverted back to YANG 1.1 if-feature statements (issue #34).

- Removed import by revisions (issue #36).

- Removed groupings only used once (issue #37).

- Removed upper-bound on hello-timeout, idle-timeout, and max-sessions (issue #38).

- Clarified that when no listen address is configured, the NETCONF/ RESTCONF server will listen on all addresses (issue #41).

- Update keep-alive reference to new section in Call Home draft (issue #42).

- Modified connection-type/persistent/keep-alives/interval-secs default value, removed the connection-type/periodic/linger-secs node, and also removed the reconnect-strategy/interval-secs node (issue #43).

- Clarified how last-connected reconnection type should work across reboots (issue #44).

- Clarified how DNS-expanded hostnames should be processed (issue #45).

- Removed text on how to implement keep-alives (now in the call-home draft) and removed the keep-alive configuration for listen connections (issue #46).

- Clarified text for .../periodic-connection/timeout-mins (issue #47).

- Fixed description on the "trusted-ca-certs" leaf-list (issue #48).

- Added optional keychain-based solution in appendix A (issue #49).

- Fixed description text for the interval-secs leaf (issue #50).

- moved idle-time into the listen, persistent, and periodic subtrees (issue #51).

- put presence statements on containers where it makes sense (issue #53).

A.8. 07 to 08

- Per WG consensus, replaced body with the keychain-based approach described in -07’s Appendix.

- Added a lot of introductory text, improved examples, and what not.

A.9. 08 to 09

- Renamed ietf-keychain to ietf-system-keychain to disambiguate from the routing area working group's keychain model (they similarly renamed their model from ietf-key-chain to ietf-routing-key-chain).
- Added an action statement to ietf-system-keychain to load a private key.
- Added a notification statement to ietf-system-keychain to notify when a certificate is nearing expiration and beyond.
- Converted all binary types to use ASN.1 DER encoding.
- Added a Design Considerations section.
- Filled in the Security Considerations section.
- Removed the Other Considerations section.
- Extended the Editorial Note section.
- Added many Normative and Informative references.

Appendix B. Open Issues

Please see: https://github.com/netconf-wg/server-model/issues.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

A. Clemm
A. Gonzalez Prieto
E. Voit
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
March 21, 2016

Subscribing to YANG datastore push updates
draft-ietf-netconf-yang-push-02.txt

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

YANG [RFC6020] was originally designed for the Netconf protocol [RFC6241], which originally put most emphasis on configuration. However, YANG is not restricted to configuration data. YANG datastores, i.e. datastores that contain data modeled according using YANG, can contain configuration as well as operational data. It is therefore reasonable to expect that data in YANG datastores will increasingly be used to support applications that are not focused on managing configurations but that are, for example, related to service assurance.

Service assurance applications typically involve monitoring operational state of networks and devices; of particular interest are changes that this data undergoes over time. Likewise, there are applications in which data and objects from one datastore need to be made available both to applications in other systems and to remote datastores [I-D.voit-netmod-yang-mount-requirements] [I-D.clemm-netmod-mount]. This requires mechanisms that allow remote systems to become quickly aware of any updates to allow to validate and maintain cross-network integrity and consistency.

Traditional approaches to remote network state visibility rely heavily on polling. With polling, data is periodically explicitly retrieved by a client from a server to stay up-to-date.

There are various issues associated with polling-based management:

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

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

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

A more effective alternative is when an application can request to be automatically updated as necessary of current content of the datastore (such as a subtree, or data in a subtree that meets a certain filter condition), and in which the server that maintains the datastore subsequently pushes those updates. However, such a solution does not currently exist.

The need to perform polling-based management is typically considered an important shortcoming of management applications that rely on MIBs polled using SNMP [RFC1157]. However, without a provision to support a push-based alternative, there is no reason to believe that management applications that operate on YANG datastores using protocols such as NETCONF or Restconf [I-D.ietf-netconf-restconf] will be any more effective, as they would follow the same request/response pattern.

While YANG allows the definition of notifications, such notifications are generally intended to indicate the occurrence of certain well-specified event conditions, such as a the onset of an alarm condition or the occurrence of an error. A capability to subscribe to and deliver event notifications has been defined in [RFC5277]. In addition, configuration change notifications have been defined in [RFC6470]. These change notifications pertain only to configuration information, not to operational state, and convey the root of the subtree to which changes were applied along with the edits, but not the modified data nodes and their values.

Accordingly, there is a need for a service that allows client applications to dynamically subscribe to updates of a YANG datastore and that allows the server to push those updates. Additionally, support for static subscriptions configured directly on the server are also useful when dynamic signaling is undesirable or unsupported. The requirements for such a service are documented in [I-D.i2rs-pub-sub-requirements]. This document proposes a solution that features the following capabilities:

- A mechanism that allows clients to dynamically subscribe to automatic datastore updates, and the means to manage those subscription. The subscription allows clients to specify which data they are interested in, and to provide optional filters with criteria that data must meet for updates to be sent. Furthermore, subscription can specify a policy that directs when updates are provided. For example, a client may request to be updated periodically in certain intervals, or whenever data changes occur.

- An alternative mechanism that allows for the static configuration of subscriptions to automatic data store updates as part of a device configuration. In addition to the aspects that are configurable for a dynamic subscription (filter criteria, update policy), static configuration of subscriptions allows for the definition of additional aspects such as intended receivers and alternative transport options.

- The ability for a server to push back on requested subscription parameters. Because not every server may support every requested interval for every piece of data, it is necessary for a server to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to negotiate subscription parameters.

- A mechanism to communicate the updates themselves. For this, the proposal leverages and extends existing YANG/Netconf/Restconf mechanisms, defining special notifications that carry updates.

This document specifies a YANG data model to manage subscriptions to data in YANG datastores, and to configure associated filters and data streams. It defines extensions to RPCs defined in [RFC5277] that allow to extend notification subscriptions to subscriptions for datastore updates. It also defines a notification that can be used to carry data updates and thus serve as push mechanism.

Editor’s note: A new draft [I-D.gonzalez-netconf-5277bis] has just been released to address many of the current Netconf Event Model’s limitations, as chartered by the Netconf Working Group. This draft will be leveraged in future revisions of this document.

2. Definitions and Acronyms

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

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

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

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

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

Dynamic subscription: A subscription negotiated between subscriber and publisher vian establish, modify, and delete RPCs respectively control plane signaling messages that are part of an existing management association between and publisher. Subscriber and receiver are the same system.

NACM: NETCONF Access Control Model

NETCONF: Network Configuration Protocol

Publisher: A server that sends push updates to a receiver according to the terms of a subscription. In general, the publisher is also the "owner" of the subscription.

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

Receiver: The target of push updates of a subscription. In case of a dynamic subscription, receiver and subscriber are the same system. However, in the case of a static subscription, the receiver may be a different system than the one that configured the subscription.

RPC: Remote Procedure Call

SNMP: Simple Network Management Protocol

Static subscription: A subscription installed as part of a configuration datastore via a configuration interface.

Subscriber: A client that negotiates a subscription with a server ("publisher"). A client that establishes a static subscription is also considered a subscriber, even if it is not necessarily the receiver of a subscription.

Subscription: A contract between a client ("subscriber") and a server ("publisher"), stipulating which information the client wishes to receive from the server (and which information the server has to provide to the client) without the need for further solicitation.

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

Subscription policy: A policy that specifies under what circumstances to push an update, e.g. whether updates are to be provided periodically or only whenever changes occur.

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

Update trigger: A trigger, as specified by a subscription policy, that causes an update to be sent, respectively a data record to be generated. An example of a trigger is a change trigger, invoked when the value of a data node changes or a data node is created or deleted, or a time trigger, invoked after the laps of a periodic time interval.

URI: Uniform Resource Identifier

YANG: A data definition language for NETCONF

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

3. Solution Overview

This document specifies a solution for a subscription service, which supports the dynamic as well as static creation of subscriptions to information updates of operational or configuration YANG data which are subsequently pushed from the server to the client.

Dynamic subscriptions are initiated by clients who want to receive push updates. Servers respond to requests for the creation of subscriptions positively or negatively. Negative responses include information about why the subscription was not accepted, in order to facilitate converging on an acceptable set of subscription parameters. Similarly, static subscriptions are configured as part of a device’s configuration. Once a subscription has been established, datastore push updates are pushed from the server to the receiver until the subscription ends.

Accordingly, the solution encompasses several components:

o The subscription model for configuration and management of the subscriptions, with a set of associated services.

o The ability to provide hints for acceptable subscription parameters, in cases where a subscription desired by a client cannot currently be served.

o The stream of push updates.

In addition, there are a number of additional considerations, such as the tie-in of the mechanisms with security mechanisms. Each of those aspects will be discussed in the following subsections.

3.1. Subscription Model

YANG-Push subscriptions are defined using a data model that is itself defined in YANG. This model is based on the subscriptions defined in [RFC5277], which are also reused in Restconf. The model is extended with several parameters, including a subscription type and a subscription ID.

The subscription model assumes the presence of a conceptual perpetual datastream "push-update" of continuous datastore updates that can be subscribed to, although other datastreams may be supported as well. A subscription refers to a datastream and specifies filters that are to be applied to, it for example, to provide only those subsets of the information that match a filter criteria. In addition, a subscription specifies a set of subscription parameters that define the trigger when data records should be sent, for example at periodic intervals or whenever underlying data items change.

The complete set of subscription parameters for both dynamic and static subscriptions is as follows:

- The stream being subscribed to. The subscription model assumes the presence of perpetual and continuous streams of updates. The stream "push-update" is always available and covers the entire set of YANG data in the server, but a system may provide other streams to choose from.

- The datastore to target. By default, the datastore will always be "running". However, it is conceivable that implementations want to also support subscriptions to updates to other datastores.

- An encoding for the data updates. By default, updates are encoded using XML, but JSON can be requested as an option and other encodings may be supported in the future.

- An optional start time for the subscription. If the specified start time is in the past, the subscription goes into effect immediately. The start time also serves as anchor time for periodic subscriptions, from which intervals at which to send updates are calculated (see also below).

- An optional stop time for the subscription. Once the stop time is reached, the subscription is automatically terminated.

- A subscription policy definition regarding the update trigger when to send new updates. The trigger can be periodic or based on change.

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

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

        + One parameter is needed to specify the dampening period, i.e. the interval that must pass before a successive update for the same data node is sent. The first time a change is detected, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed, containing the value of the data node that is then valid.

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

        + A third parameter specifies whether or not a complete update with all the subscribed data should be sent at the beginning of a subscription.

- Optionally, a filter, or set of filters, describing the subset of data items in the stream’s data records that are of interest to the subscriber. The server should only send to the subscriber the data items that match the filter(s), when present. The absence of a filter indicates that all data items from the stream are of interest to the subscriber and all data records must be sent in their entirety to the subscriber. Three types of filters are supported: subtree filters, with the same semantics as defined in [RFC 6241], XPath filters, and RFC 5277 filter, with the same semantics as defined in [RFC 5277]. Additional filter types can be added through augmentations. Filters can be specified "inline" as part of the subscription, or can be configured separately and referenced by a subscription, in order to facilitate reuse of complex filters.

In addition, for the configuration of static subscriptions, the following parameters are supported:

- One or more receiver IP addresses (and corresponding ports) intended as the destination for push updates for each subscription. In addition the transport protocol for each destination may be defined.

- Optional parameters to identify an egress interface or IP address / VRF where a subscription updates should be pushed from the publisher.

The subscription data model is specified as part of the YANG data model described later in this specification. Specifically, the subscription parameters are defined in the "subscription-info" and "update-policy" groupings. Receiver information is defined in the "receiver-info" grouping. Information about the source address is defined in the "push-source-info" grouping. It is conceivable that additional subscription parameters might be added in the future. This can be accomplished through augmentation of the subscription data model.

3.2. Negotiation of Subscription Policies

A subscription rejection can be caused by the inability of the server to provide a stream with the requested semantics. For example, a server may not be able to support "on-change" updates for operational data, or only support them for a limited set of data nodes. Likewise, a server may not be able to support a requested updated frequency, or a requested encoding.

YANG-Push supports a simple negotiation between clients and servers for subscription parameters. The negotiation is limited to a single pair of subscription request and response. For negative responses, the server SHOULD include in the returned error what subscription parameters would have been accepted for the request. The returned acceptable parameters constitute suggestions that, when followed, increase the likelihood of success for subsequent requests. However, they are no guarantee that subsequent requests for this client or others will in fact be accepted.

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

3.3. On-Change Considerations

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

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

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

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

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

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

3.4. Data Encodings

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

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

3.4.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been simply retrieved using a get operation and is encoded in the same way. XML encoding rules for data nodes are defined in [RFC6020]. JSON encoding rules are defined in [I-D.ietf-netmod-yang-json]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

3.4.2. On-Change Subscriptions

In an on-change subscription, updates need to allow to differentiate between data nodes that were newly created since the last update, data nodes that were deleted, and data nodes whose value changed.

XML encoding rules correspond to how data would be encoded in input to Netconf edit-config operations as specified in [RFC6241] section 7.2, adding "operation" attributes to elements in the data subtree. Specifically, the following values will be utilized:

- create: The data identified by the element has been added since the last update.

- delete: The data identified by the element has been deleted since the last update.

- merge: The data identified by the element has been changed since the last update.

- replace: The data identified by the element has been replaced with the update contents since the last update.

The remove value will not be utilized.

Contrary to edit-config operations, the data is sent from the server to the client, not from the client to the server, and will not be restricted to configuration data.

JSON encoding rules are roughly analogous to how data would be encoded in input to a YANG-patch operation, as specified in [I-D.ietf-netconf-yang-patch] section 2.2. However, no edit-ids will be needed. Specifically, changes will be grouped under respective "operation" containers for creations, deletions, and modifications.

3.5. Subscription Filters

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

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

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

- RFC5277 filter: A filter that allows for matching of update notification records per RFC 5277.

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

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

3.6. Subscription State Model at the Publisher

Below is the state machine for the publisher. It is important to note that a subscription doesn’t exist at the publisher until it is accepted and made active. The mere request by a subscriber to establish a subscription is insufficient for that asserted subscription to be externally visible via this state machine.

```
                          .-------.
                       | start |
                       '-------'
                           |
                        establish
                           |
                           |   .----------modify-------------.lex
                           v   v                              '
                     .-----------.                      .-----------.
          .--------. |           |------>suspend------->|           |
        modify      '|  active   |                      | suspended |
          '--------->|           |<----resume----<------|           |
                     '-----------'                      '-----------'
                           |                                  |
                        delete                             delete
                           |                                  |
                           v                                  |
                       .-------.                              |
                       |  end  |<-----------------------------'
                       '-------'
```

Figure 1: Subscription states at publisher

Of interest in this state machine are the following:

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

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

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

3.7. Push Data Stream and Transport Mapping

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

A more suitable mechanism to consider is therefore that of a notification. There are however some specifics that need to be considered. Contrary to other notifications that are associated with alarms and unexpected event occurrences, push updates are solicited, i.e. tied to a particular subscription which triggered the notification, and arguably only of interest to the subscriber, respectively the intended receiver of the subscription. A subscription therefore needs to be able to distinguish between streams that underlie push updates and streams of other notifications. By the same token, notifications associated with updates and subscriptions to updates need to be distinguished from other notifications, in that they enter a datastream of push updates, not a stream of other event notifications.

A push update notification contains several parameters:

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

o A data node that contains a representation of the datastore subtree containing the updates. The subtree is filtered per access control rules to contain only data that the subscriber is authorized to see. Also, depending on the subscription type, i.e., specifically for on-change subscriptions, the subtree contains only the data nodes that contain actual changes. (This can be simply a node of type string or, for XML-based encoding, anyxml.)

Notifications are sent using <notification> elements as defined in [RFC5277]. Alternative transports are conceivable but outside the scope of this specification.

The solution specified in this document uses notifications to define datastore updates. The contents of the notification includes a set of explicitly defined data nodes. For this purpose, two new generic notifications are introduced, "push-update" and "push-change-update". Those notifications define how mechanisms that carry YANG notifications (e.g. Netconf notifications and Restconf) can be used to carry data records with updates of datastore contents as specified by a subscription. It is possible also map notifications to other transports and encodings and use the same subscription model; however, the definition of such mappings is outside the scope of this document.

Push-update notification defines updates for a periodic subscription, as well as for the initial update of an on-change subscription used to synchronize the receiver at the start of a new subscription. The update record contains a data snippet that contains an instantiated subtree with 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 e.g. a Netconf "get"-operation, with the same filters applied.

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

Push-change-update notification defines updates for on-change subscriptions. The update record here contains a data snippet that indicates the changes that data nodes have undergone, i.e. that indicates which data nodes have been created, deleted, or had changes to their values. The format follows the same format that operations that apply changes to a data tree would apply, indicating the creates, deletes, and modifications of data nodes.

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

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

Figure 2: Push example

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

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

Figure 3: Push example for on change

The equivalent update when requesting json encoding:

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

Figure 4: Push example for on change with JSON

When the beta leaf is deleted, the server may send

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

Figure 5: 2nd push example for on change update

3.8. Subscription management

There are two ways in which subscriptions can be managed: RPC-based and configuration based. Any given subscription is either RPC-based or configuration-based. There is no mixing-and-matching of RPC and configuration operations. Specifically, a configured subscription cannot be modified or deleted using RPC. Likewise, a subscription established via RPC cannot be modified or deleted through configuration operations.

Note: In order to distinguish subscribing to a stream of YANG-Push updates from subscribing to a regular event stream and avoid confusion with the <creat-subscription> RPC defined in RFC 5277, the term "establish-subscription" is used instead of "create-subscription".

3.8.1. Subscription management by RPC

RPC-based subscription allows a subscriber to establish a subscription via an RPC call. The subscriber and the receiver are the same entity, i.e. a subscriber cannot subscribe or in other ways interfere with a subscription on another receiver’s behalf. The lifecycle of the subscription is dependent on the lifecyle of the transport session over which the subscription was requested. For example, when a Netconf session over which a subscription was established is torn down, the subscription is automatically terminated (and needs to be re-initiated when a new session is established). Alternatively, a subscriber can also decide to delete a subscription via another RPC.

When an establish-subscription request is successful, the subscription identifier of the freshly established subscription is returned.

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

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

For instance, for the following request:

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

Figure 6: Establish-Subscription example

the server might return:

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

Figure 7: Error response example

A subscriber that establishes a subscription using RPC can modify or delete the subscription using other RPCs. When the session between subscriber and publisher is terminated, the subscription is implicitly deleted.

3.8.2. Subscription management by configuration

Configuration-based subscription allows a subscription to be established as part of a server’s configuration. This allows to persist subscriptions. Persisted subscriptions allow for a number of additional options than RPC-based subscriptions. As part of a configured subscription, a receiver needs to be specified. It is thus possible to have a different system acting as subscriber (the client creating the subscription) and as receiver (the client receiving the updates). In addition, a configured subscription allows to specify which transport protocol should be used, as well as the sender source (for example, a particular interface or an address of a specific VRF) from which updates are to be pushed.

Configuration-based subscriptions cannot be modified or deleted using RPCs. Instead, configured subscriptions are deleted as part of regular configuration operations. Servers SHOULD reject attempts to modify configurations of active subscriptions. This way, race conditions in which a receiver may not be aware of changed subscription policies are avoided.

3.9. Other considerations

3.9.1. Authorization

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

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

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

```
                 +-------------+                 +-------------+
    subscription |  protocol   |                 |   target    |
    request -->  |  operation  | ------------->  |  data node  |
                 |  allowed?   |   datastore     |   access    |
                 +-------------+   or state      |  allowed?   |
                                   data access   +-------------+
```

Figure 8: Access control for subscription

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

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

```
                   +-------------+      +-------------------+
    subscription   |  data node  |  yes |                   |
    update  -->    |   access    | ---> | add data node     |
                   |  allowed?   |      | to update message |
                   +-------------+      +-------------------+
```

Figure 9: Access control for push updates

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

3.9.2. Additional subscription primitives

Other possible operations include the ability for a Subscriber to request the suspension/resumption of a Subscription with a Publisher. However, subscriber driven suspension is not viewed as essential at this time, as a simpler alternative is to remove a subscription and reestablish it when needed.

It should be noted that this does not affect the ability of the Publisher to suspend a subscription. This can occur in cases the server is not able to serve the subscription for a certain period of time, and indicated by a corresponding notification.

3.9.3. Robustness and reliability considerations

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

YANG-Push uses a secure and reliable transport. Notifications are not getting reordered, and in addition contain a time stamp. For those reasons, for the transport of push-updates, we believe that additional reliability mechanisms at the application level, such as sequence numbers for push updates, are not required.

At the same time, it is conceivable that under certain circumstances, a push server is not able to generate the update notifications that it had committed to when accepting a subcription. In those circumstances, the server needs to inform the receiver of the situation. For this purpose, notifications are defined that a push server can use to inform subscribers/ receivers when a subscription is (temporarily) suspended, when a suspended subscription is resumed, and when a a subscription is terminated. This way, receivers will be able to rely on a subscription, knowing that they will be informed of any situations in which updates might be missed.

3.9.4. Update size and fragmentation considerations

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

3.9.5. Push data streams

There are several conceptual data streams introduced in this specification:

- yang-push includes the entirety of YANG data, including both configuration and operational data.

- operational-push includes all operational (read-only) YANG data

- config-push includes all YANG configuration data.

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

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

- other custom datastreams

Those data streams make particular sense for use cases involving service assurance (not relying on operational data), and for use cases requiring on-change update triggers which make no sense to support in conjunction with fast-changing counters. While it is possible to specify subtree filters on yang-push to the same effect, having those data streams greatly simplifies articulating subscriptions in such scenarios.

3.9.6. Implementation considerations

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

Even periodic retrieval of operational state alone, to be able to push it, can consume considerable system resources. Configuration data may in many cases be persisted in an actual database or a configuration file, where retrieval of the database content or the file itself is reasonably straightforward and computationally inexpensive. However, retrieval of operational data may, depending on the implementation, require invocation of APIs, possibly on an object-by-object basis, possibly involving additional internal interrupts, etc.

For those reasons, if is important for an implementation to understand what subscriptions it can or cannot support. It is far preferrable to decline a subscription request, than to accept it only to result in subsequent failure later.

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

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

3.9.7. Alignment with RFC 5277bis

A new draft has been chartered by the Netconf Working Group to replace the current Netconf Event Model defined in RFC 5277. Future revisions of this document will leverage RFC 5277bis as applicable. It is anticipated that portions of the data model and subscription management that are now defined in this this document and that are not applicable only to YANG-Push, but to more general event subscriptions, will move to RFC 5277bis.

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.

```
module: ietf-yang-push
   +--ro update-streams
   |  +--ro update-stream*   update-stream
   +--rw filters
   |  +--rw filter* [filter-id]
   |     +--rw filter-id         filter-id
   |     +--rw (filter-type)?
   |        +--:(subtree)
   |        |  +--rw subtree-filter
   |        +--:(xpath)
   |        |  +--rw xpath-filter?     yang:xpath1.0
   |        +--:(rfc5277)
   |           +--rw filter
   +--rw subscription-config {configured-subscriptions}?
   |  +--rw yang-push-subscription* [subscription-id]
   |     +--rw subscription-id            subscription-id
   |     +--rw stream?                    update-stream
   |     +--rw encoding?                  encoding
   |     +--rw subscription-start-time?   yang:date-and-time
   |     +--rw subscription-stop-time?    yang:date-and-time
   |     +--rw (filterspec)?
   |     |  +--:(inline)
   |     |  |  +--rw (filter-type)?
   |     |  |     +--:(subtree)
   |     |  |     |  +--rw subtree-filter
   |     |  |     +--:(xpath)
   |     |  |     |  +--rw xpath-filter?              yang:xpath1.0
   |     |  |     +--:(rfc5277)
   |     |  |        +--rw filter
   |     |  +--:(by-reference)
   |     |     +--rw filter-ref?                filter-ref
   |     +--rw (update-trigger)?
   |     |  +--:(periodic)
   |     |  |  +--rw period                     yang:timeticks
   |     |  +--:(on-change) {on-change}?
   |     |     +--rw no-synch-on-start?         empty
   |     |     +--rw dampening-period           yang:timeticks
   |     |     +--rw excluded-change*           change-type
   |     +--rw receiver* [address]
   |     |  +--rw address     inet:host
   |     |  +--rw port?       inet:port-number
   |     |  +--rw protocol?   transport-protocol
   |     +--rw (push-source)?
   |     |  +--:(interface-originated)
   |     |  |  +--rw source-interface?         if:interface-ref
   |     |  +--:(address-originated)
   |     |     +--rw source-vrf?                uint32
   |     |     +--rw source-address             inet:ip-address-no-zone
   |     +--rw dscp?               inet:dscp {configured-subscriptions}?
   |     +--rw subscription-priority?     uint8
   |     +--rw subscription-dependency?   string
   +--ro subscriptions
      +--ro yang-push-subscription* [subscription-id]
         +--ro subscription-id            subscription-id
         +--ro configured-subscription?empty {configured-subscriptions}?
         +--ro subscription-status?       identityref
         +--ro stream?                    update-stream
         +--ro encoding?                  encoding
         +--ro subscription-start-time?   yang:date-and-time
         +--ro subscription-stop-time?    yang:date-and-time
         +--ro (filterspec)?
         |  +--:(inline)
         |  |  +--ro (filter-type)?
         |  |     +--:(subtree)
         |  |     |  +--ro subtree-filter
         |  |     +--:(xpath)
         |  |     |  +--ro xpath-filter?              yang:xpath1.0
         |  |     +--:(rfc5277)
         |  |        +--ro filter
         |  +--:(by-reference)
         |     +--ro filter-ref?                filter-ref
         +--ro (update-trigger)?
         |  +--:(periodic)
         |  |  +--ro period                     yang:timeticks
         |  +--:(on-change) {on-change}?
         |     +--ro no-synch-on-start?         empty
         |     +--ro dampening-period           yang:timeticks
         |     +--ro excluded-change*           change-type
         +--ro receiver* [address]
         |  +--ro address     inet:host
         |  +--ro port?       inet:port-number
         |  +--ro protocol?   transport-protocol
         +--ro (push-source)?
         |  +--:(interface-originated)
         |  |  +--ro source-interface?          if:interface-ref
         |  +--:(address-originated)
         |     +--ro source-vrf?                uint32
         |     +--ro source-address             inet:ip-address-no-zone
         +--ro dscp?               inet:dscp {configured-subscriptions}?
         +--ro subscription-priority?     uint8
         +--ro subscription-dependency?   string
rpcs:
   +---x establish-subscription
   |  +---w input
   |  |  +---w stream?                    update-stream
   |  |  +---w encoding?                  encoding
   |  |  +---w subscription-start-time?   yang:date-and-time
   |  |  +---w subscription-stop-time?    yang:date-and-time
   |  |  +---w (filterspec)?
   |  |  |  +--:(inline)
   |  |  |  |  +---w (filter-type)?
   |  |  |  |     +--:(subtree)
   |  |  |  |     |  +---w subtree-filter
   |  |  |  |     +--:(xpath)
   |  |  |  |     |  +---w xpath-filter?             yang:xpath1.0
   |  |  |  |     +--:(rfc5277)
   |  |  |  |        +---w filter
   |  |  |  +--:(by-reference)
   |  |  |     +---w filter-ref?                filter-ref
   |  |  +---w (update-trigger)?
   |  |  |  +--:(periodic)
   |  |  |  |  +---w period                     yang:timeticks
   |  |  |  +--:(on-change) {on-change}?
   |  |  |     +---w no-synch-on-start?         empty
   |  |  |     +---w dampening-period           yang:timeticks
   |  |  |     +---w excluded-change*           change-type
   |  |  +---w dscp?               inet:dscp {configured-subscriptions}?
   |  |  +---w subscription-priority?     uint8
   |  |  +---w subscription-dependency?   string
   |  +--ro output
   |     +--ro subscription-result        subscription-result
   |     +--ro (result)?
   |        +--:(success)
   |        |  +--ro subscription-id            subscription-id
   |        +--:(no-success)
   |           +--ro stream?                    update-stream
   |           +--ro encoding?                  encoding
   |           +--ro subscription-start-time?   yang:date-and-time
   |           +--ro subscription-stop-time?    yang:date-and-time
   |           +--ro (filterspec)?
   |           |  +--:(inline)
   |           |  |  +--ro (filter-type)?
   |           |  |     +--:(subtree)
   |           |  |     |  +--ro subtree-filter
   |           |  |     +--:(xpath)
   |           |  |     |  +--ro xpath-filter?            yang:xpath1.0
   |           |  |     +--:(rfc5277)
   |           |  |        +--ro filter
   |           |  +--:(by-reference)
   |           |     +--ro filter-ref?                filter-ref
   |           +--ro (update-trigger)?
   |           |  +--:(periodic)
   |           |  |  +--ro period                     yang:timeticks
   |           |  +--:(on-change) {on-change}?
   |           |     +--ro no-synch-on-start?         empty
   |           |     +--ro dampening-period           yang:timeticks
   |           |     +--ro excluded-change*           change-type
   |           +--ro dscp?         inet:dscp {configured-subscriptions}?
   |           +--ro subscription-priority?     uint8
   |           +--ro subscription-dependency?   string
   +---x modify-subscription
   |  +---w input
   |  |  +---w subscription-id?           subscription-id
   |  |  +---w stream?                    update-stream
   |  |  +---w encoding?                  encoding
   |  |  +---w subscription-start-time?   yang:date-and-time
   |  |  +---w subscription-stop-time?    yang:date-and-time
   |  |  +---w (filterspec)?
   |  |  |  +--:(inline)
   |  |  |  |  +---w (filter-type)?
   |  |  |  |     +--:(subtree)
   |  |  |  |     |  +---w subtree-filter
   |  |  |  |     +--:(xpath)
   |  |  |  |     |  +---w xpath-filter?              yang:xpath1.0
   |  |  |  |     +--:(rfc5277)
   |  |  |  |        +---w filter
   |  |  |  +--:(by-reference)
   |  |  |     +---w filter-ref?                filter-ref
   |  |  +---w (update-trigger)?
   |  |  |  +--:(periodic)
   |  |  |  |  +---w period                     yang:timeticks
   |  |  |  +--:(on-change) {on-change}?
   |  |  |     +---w no-synch-on-start?         empty
   |  |  |     +---w dampening-period           yang:timeticks
   |  |  |     +---w excluded-change*           change-type
   |  |  +---w dscp?               inet:dscp {configured-subscriptions}?
   |  |  +---w subscription-priority?     uint8
   |  |  +---w subscription-dependency?   string
   |  +--ro output
   |     +--ro subscription-result        subscription-result
   |     +--ro stream?                    update-stream
   |     +--ro encoding?                  encoding
   |     +--ro subscription-start-time?   yang:date-and-time
   |     +--ro subscription-stop-time?    yang:date-and-time
   |     +--ro (filterspec)?
   |     |  +--:(inline)
   |     |  |  +--ro (filter-type)?
   |     |  |     +--:(subtree)
   |     |  |     |  +--ro subtree-filter
   |     |  |     +--:(xpath)
   |     |  |     |  +--ro xpath-filter?             yang:xpath1.0
   |     |  |     +--:(rfc5277)
   |     |  |        +--ro filter
   |     |  +--:(by-reference)
   |     |     +--ro filter-ref?                filter-ref
   |     +--ro (update-trigger)?
   |     |  +--:(periodic)
   |     |  |  +--ro period                     yang:timeticks
   |     |  +--:(on-change) {on-change}?
   |     |     +--ro no-synch-on-start?         empty
   |     |     +--ro dampening-period           yang:timeticks
   |     |     +--ro excluded-change*           change-type
   |     +--ro dscp?               inet:dscp {configured-subscriptions}?
   |     +--ro subscription-priority?     uint8
   |     +--ro subscription-dependency?   string
   +---x delete-subscription
      +---w input
         +---w subscription-id?   subscription-id
notifications:
   +---n push-update
   |  +--ro subscription-id            subscription-id
   |  +--ro time-of-update?            yang:date-and-time
   |  +--ro (encoding)?
   |     +--:(encode-xml)
   |     |  +--ro datastore-contents-xml?    datastore-contents-xml
   |     +--:(encode-json) {json}?
   |        +--ro datastore-contents-json?   datastore-contents-json
   +---n push-change-update {on-change}?
   |  +--ro subscription-id           subscription-id
   |  +--ro time-of-update?           yang:date-and-time
   |  +--ro (encoding)?
   |     +--:(encode-xml)
   |     |  +--ro datastore-changes-xml?    datastore-changes-xml
   |     +--:(encode-json) {json}?
   |        +--ro datastore-changes-yang?   datastore-changes-json
   +---n subscription-started
   |  +--ro subscription-id            subscription-id
   |  +--ro stream?                    update-stream
   |  +--ro encoding?                  encoding
   |  +--ro subscription-start-time?   yang:date-and-time
   |  +--ro subscription-stop-time?    yang:date-and-time
   |  +--ro (filterspec)?
   |  |  +--:(inline)
   |  |  |  +--ro (filter-type)?
   |  |  |     +--:(subtree)
   |  |  |     |  +--ro subtree-filter
   |  |  |     +--:(xpath)
   |  |  |     |  +--ro xpath-filter?             yang:xpath1.0
   |  |  |     +--:(rfc5277)
   |  |  |        +--ro filter
   |  |  +--:(by-reference)
   |  |     +--ro filter-ref?                filter-ref
   |  +--ro (update-trigger)?
   |  |  +--:(periodic)
   |  |  |  +--ro period                     yang:timeticks
   |  |  +--:(on-change) {on-change}?
   |  |     +--ro no-synch-on-start?         empty
   |  |     +--ro dampening-period           yang:timeticks
   |  |     +--ro excluded-change*           change-type
   |  +--ro dscp?                 inet:dscp {configured-subscriptions}?
   |  +--ro subscription-priority?     uint8
   |  +--ro subscription-dependency?   string
   +---n subscription-suspended
   |  +--ro subscription-id    subscription-id
   |  +--ro reason?            subscription-susp-reason
   +---n subscription-resumed
   |  +--ro subscription-id    subscription-id
   +---n subscription-modified
   |  +--ro subscription-id            subscription-id
   |  +--ro stream?                    update-stream
   |  +--ro encoding?                  encoding
   |  +--ro subscription-start-time?   yang:date-and-time
   |  +--ro subscription-stop-time?    yang:date-and-time
   |  +--ro (filterspec)?
   |  |  +--:(inline)
   |  |  |  +--ro (filter-type)?
   |  |  |     +--:(subtree)
   |  |  |     |  +--ro subtree-filter
   |  |  |     +--:(xpath)
   |  |  |     |  +--ro xpath-filter?              yang:xpath1.0
   |  |  |     +--:(rfc5277)
   |  |  |        +--ro filter
   |  |  +--:(by-reference)
   |  |     +--ro filter-ref?                filter-ref
   |  +--ro (update-trigger)?
   |  |  +--:(periodic)
   |  |  |  +--ro period                     yang:timeticks
   |  |  +--:(on-change) {on-change}?
   |  |     +--ro no-synch-on-start?         empty
   |  |     +--ro dampening-period           yang:timeticks
   |  |     +--ro excluded-change*           change-type
   |  +--ro dscp?                  inet:dscp {configured-subscriptions}?
   |  +--ro subscription-priority?     uint8
   |  +--ro subscription-dependency?   string
   +---n subscription-terminated
      +--ro subscription-id    subscription-id
      +--ro reason?            subscription-term-reason
```

Figure 10: Model structure

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

4.2. Update streams

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

4.3. Filters

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

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

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

4.4. Subscription configuration

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

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

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

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

- "subscription-start-time" specifies when the subscription is supposed to start. The start time also serves as anchor time for periodic subscriptions (see below).

- "subscription-stop-time" specifies a stop time for the subscription. Once the stop time is reached, the subscription is automatically terminated. However, even when terminated, the subscription entry remains part of the configuration unless explicity deleted from the configuration. It is possible to effectively "resume" a stopped subscription by reconfiguring the stop time.

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

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

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

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

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

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

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

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

4.5. Subscription monitoring

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

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

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

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

4.6. Notifications

A server needs to indicate any changes in status of a subscription to the receiver through a notification. Specifically, subscribers need to be informed of the following:

- A subscription has been temporarily suspended (including the reason)

- A subscription (that had been suspended earlier) is once again operational

- A subscription has been terminated (including the reason)

- A subscription has been modified (including the current set of subscription parameters in effect)

Finally, a server might provide additional information about subscriptions, such as statistics about the number of data updates that were sent. However, such information is currently outside the scope of this specification.

4.7. RPCs

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

4.7.1. Establish-subscription RPC

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

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

Figure 11: Establish-subscription RPC

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

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

Figure 12: Establish-subscription positive RPC response

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

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

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

Figure 13: Establish-subscription access denied response

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

For example, for the following request:

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

Figure 14: Establish-subscription request example 2

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

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

Figure 15: Establish-subscription error response example 2

4.7.2. Modify-subscription RPC

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

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

Figure 16: Modify subscription request

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

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

Figure 17: Modify subscription response

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

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

4.7.3. Delete-subscription RPC

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

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

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

Figure 18: Delete subscription

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

5. YANG module

```
<CODE BEGINS> file "ietf-yang-push@2016-03-21.yang"
module ietf-yang-push {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import ietf-interfaces {
    prefix if;
  }

  organization "IETF";
  contact
```

```
     "WG Web:   <http://tools.ietf.org/wg/netconf/>
      WG List:  <mailto:netconf@ietf.org>

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

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

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

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

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

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

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

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

   revision 2016-03-21 {
     description
       "Changes to grouping structure, RPC definitions, filter
        definitions, origin interface and receiver definitions.";
     reference "YANG Datastore Push, draft-ietf-netconf-yang-push-02";
   }

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

   feature json {
     description
       "This feature indicates that JSON encoding of push updates
        is supported.";
   }

   feature configured-subscriptions {
     description
       "This feature indicates that management plane configuration
        of subscription is supported.";
   }

   identity subscription-result {
     description
       "Base identity for RPC responses to requests surrounding
        management (e.g. creation, modification) of
        subscriptions.";
   }

   identity ok {
     base subscription-result;
     description
       "OK - RPC was successful and was performed as requested.";
   }

   identity error {
     base subscription-result;
     description
       "RPC was not successful.
        Base identity for error return codes.";
   }

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

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

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

   identity error-data-not-authorized {
     base error;
     description
```

```
        "No read authorization for a requested data node.";
    }

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

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

    identity subscription-stream-status {
      description
        "Base identity for the status of subscriptions and
         datastreams.";
    }

    identity active {
      base subscription-stream-status;
      description
        "Status is active and healthy.";
    }

    identity inactive {
      base subscription-stream-status;
      description
        "Status is inactive, for example outside the
         interval between start time and stop time.";
    }

    identity suspended {
      base subscription-stream-status;
      description
        "The status is suspended, meaning that the push server
         is currently unable to provide the negotiated updates
         for the subscription.";
    }

    identity subscription-errors {
      description
        "Base identity for subscription error status.
         This identity is not to be confused with error return
         codes for RPCs";
```

```
  }

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

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

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

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

  identity event-stream {
    description
      "Base identity to represent a generic stream of event
       notifications.";
  }

  identity update-stream {
    base event-stream;
    description
      "Base identity to represent a conceptual system-provided
       datastream of datastore updates with predefined semantics.";
  }

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

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

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

  identity custom-stream {
    base update-stream;
    description
      "A category of customizable datastream for datastore
       updates with contents that have defined by a user.";
  }

  identity netconf-stream {
    base event-stream;
    description
      "Default notification stream";
  }

  identity encodings {
    description
      "Base identity to represent data encodings";
  }

  identity encode-xml {
    base encodings;
    description
      "Encode data using XML";
  }

  identity encode-json {
    base encodings;
    description
      "Encode data using JSON";
  }

  identity transport {
    description
   "An identity that represents a transport protocol for event updates";
  }
  identity netconf {
    base transport;
    description
      "Netconf notifications as a transport";
  }

  identity restconf {
    base transport;
    description
      "Restconf notifications as a transport";
  }

  typedef datastore-contents-xml {
    type string;
    description
      "This type is be used to represent datastore contents,
       i.e. a set of data nodes with their values, in XML.
       The syntax corresponds to the syntax of the data payload
       returned in a corresponding Netconf get operation with the
       same filter parameters applied.";
    reference "RFC 6241 section 7.7";
  }

  typedef datastore-changes-xml {
    type string;
    description
      "This type is used to represent a set of changes in a
       datastore encoded in XML, indicating for datanodes whether
       they have been created, deleted, or updated.  The syntax
       corresponds to the syntax used to when editing a
       datastore using the edit-config operation in Netconf.";
    reference "RFC 6241 section 7.2";
  }

  typedef datastore-contents-json {
    type string;
    description
      "This type is be used to represent datastore contents,
       i.e. a set of data nodes with their values, in JSON.
       The syntax corresponds to the syntax of the data
       payload returned in a corresponding RESTCONF get
       operation with the same filter parameters applied.";
    reference "RESTCONF Protocol";
  }

  typedef datastore-changes-json {
    type string;
    description
```

```
      "This type is used to represent a set of changes in a
       datastore encoded in JSON, indicating for datanodes whether
       they have been created, deleted, or updated.  The syntax
       corresponds to the syntax used to patch a datastore
       using the yang-patch operation with Restconf.";
    reference "draft-ietf-netconf-yang-patch";
  }

  typedef subscription-id {
    type uint32;
    description
      "A type for subscription identifiers.";
  }

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

  typedef subscription-result {
    type identityref {
      base subscription-result;
    }
    description
      "The result of a subscription operation";
  }

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

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

  typedef encoding {
    type identityref {
      base encodings;
    }
    description
      "Specifies a data encoding, e.g. for a data subscription.";
  }

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

  typedef transport-protocol {
    type identityref {
      base transport;
    }
    description
      "Specifies transport protocol used to send updates to a
       receiver.";
  }

  typedef push-source {
    type enumeration {
      enum "interface-originated" {
        description
          "Pushes will be sent from a specific interface on a
           Publisher";
      }
      enum "address-originated" {
        description
          "Pushes will be sent from a specific address on a
           Publisher";
      }
    }
    description
```

```
        "Specifies from where objects will be sourced when being pushed
         off a publisher.";
    }

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

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

    grouping datatree-filter {
      description
        "This grouping defines filters for a datastore tree.";
      choice filter-type {
        description
          "A filter needs to be a single filter of a given type.
           Mixing and matching of multiple filters does not occur
           at the level of this grouping.";
        case subtree {
          description
            "Subtree filter.";
          anyxml subtree-filter {
            description
              "Subtree-filter used to specify the data nodes targeted
               for subscription within a subtree, or subtrees, of a
               conceptual YANG datastore.
               It may include additional criteria,
               allowing users to receive only updates of a limited
               set of data nodes that match those filter criteria.
               This will be used to define what
               updates to include in a stream of update events, i.e.
               to specify for which data nodes update events should be
               generated and specific match expressions that objects
               need to meet. The syntax follows the subtree filter
               syntax specified in RFC 6241, section 6.";
            reference "RFC 6241 section 6";
          }
        }
        case xpath {
          description
            "XPath filter";
          leaf xpath-filter {
            type yang:xpath1.0;
            description
              "Xpath defining the data items of interest.";
          }
        }
        case rfc5277 {
          anyxml filter {
            description
              "Subtree filter per RFC 5277";
          }
        }
      }
    }

    grouping update-policy {
      description
        "This grouping describes the conditions under which an
         update will be sent as part of an update stream.";
      choice update-trigger {
        description
          "Defines necessary conditions for sending an event to
           the subscriber.";
        case periodic {
          description
            "The agent is requested to notify periodically the
             current values of the datastore or the subset
             defined by the filter.";
          leaf period {
            type yang:timeticks;
            mandatory true;
            description
              "Duraton of time which should occur between periodic
               push updates.  Where the anchor of a start-time is
               available, the push will include the objects and their
               values which exist at an exact multiple of timeticks
               aligning to this start-time anchor.";
          }
        }
        case on-change {
          if-feature "on-change";
          description
            "The agent is requested to notify changes in
             values in the datastore or a subset of it defined
             by a filter.";
          leaf no-synch-on-start {
            type empty;
            description
              "This leaf acts as a flag that determines behavior at the
               start of the subscription.  When present,
               synchronization of state at the beginning of the
               subscription is outside the scope of the subscription.
               Only updates about changes that are observed from the
               start time, i.e. only push-change-update notifications
               are sent.
               When absent (default behavior), in order to facilitate
               a receiver’s synchronization, a full update is sent
               when the subscription starts using a push-update
               notification, just like in the case of a periodic
               subscription.  After that, push-change-update
               notifications are sent.";
          }
          leaf dampening-period {
            type yang:timeticks;
            mandatory true;
            description
              "Minimum amount of time that needs to have
               passed since the last time an update was
               provided.";
          }
          leaf-list excluded-change {
            type change-type;
            description
              "Use to restrict which changes trigger an update.
               For example, if modify is excluded, only creation and
               deletion of objects is reported.";
          }
        }
      }
    }

    grouping subscription-info {
      description
        "This grouping describes basic information concerning a
         subscription.";
      leaf stream {
        type update-stream;
        description
          "The stream being subscribed to.";
      }
      leaf encoding {
        type encoding;
        default "encode-xml";
        description
          "The type of encoding for the subscribed data.
           Default is XML";
      }
      leaf subscription-start-time {
        type yang:date-and-time;
        description
          "Designates the time at which a subscription is supposed
           to start, or immediately, in case the start-time is in
           the past. For periodic subscription, the start time also
           serves as anchor time from which the time of the next
           update is computed. The next update will take place at the
           next period interval from the anchor time.
           For example, for an anchor time at the top of a minute
           and a period interval of a minute, the next update will
           be sent at the top of the next minute.";
      }
      leaf subscription-stop-time {
        type yang:date-and-time;
        description
          "Designates the time at which a subscription will end.
           When a subscription reaches its stop time, it will be
           automatically deleted. No final push is required unless there
           is exact alignment with the end of a periodic subscription
           period.";
      }
      choice filterspec {
        description
          "The filter to be applied to the stream as part of the
           subscription.  The filter defines which updates of the
           data stream are of interest to a subscriber.
           The filter can be specified in-line
           or configured separately and referenced here.
           If no filter is specified, the entire datatree
           is of interest.";
        case inline {
          description
            "Filter is defined as part of the subscription.";
          uses datatree-filter;
        }
        case by-reference {
          description
            "Incorporate a filter that has been configured
             separately.";
          leaf filter-ref {
            type filter-ref;
            description
              "References filter which is associated with the
```

```
             subscription.";
        }
      }
    }
  }

  grouping push-source-info {
    description
      "Defines the sender source from which push updates
       for a configured subscription are pushed.";
    choice push-source {
      description
        "Identifies the egress interface on the Publisher from
         which pushed updates will or are being sent.";
      case interface-originated {
        description
          "When the push source is out of an interface on the
           Publisher established via static configuration.";
        leaf source-interface {
          type if:interface-ref;
          description
            "References the interface for pushed updates.";
        }
      }
      case address-originated {
        description
          "When the push source is out of an IP address on the
           Publisher established via static configuration.";
        leaf source-vrf {
          type uint32 {
            range "16..1048574";
          }
          description
            "Label of the vrf.";
        }
        leaf source-address {
          type inet:ip-address-no-zone;
          mandatory true;
          description
            "The source address for the pushed objects.";
        }
      }
    }
  }

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

```
       configured subscription.  This includes
       specifying the receiver, as well as defining
       any network and transport aspects when pushing of
       updates occurs outside of Netconf or Restconf.";
    list receiver {
      key "address";
      description
        "A single host or multipoint address intended as a target
         for the pushed updates for a subscription.";
      leaf address {
        type inet:host;
        description
          "Specifies the address for the traffic to reach a
           remote host. One of the following must be
           specified: an ipv4 address, an ipv6 address,
           or a host name.";
      }
      leaf port {
        type inet:port-number;
        description
          "This leaf specifies the port number to use for messages
           destined for a receiver.";
      }
      leaf protocol {
        type transport-protocol;
        default "netconf";
        description
          "This leaf specifies the transport protocol used
           to deliver messages destined for the receiver.";
      }
    }
  }

 grouping subscription-qos {
    description
     "This grouping describes Quality of Service information
      concerning a subscription.  This information is passed to lower
      layers for transport priortization and treatment";
    leaf dscp {
      if-feature configured-subscriptions;
      type inet:dscp;
      default 0;
      description
        "The push update’s IP packet transport priority.
        This is made visible across network hops to receiver.
        The transport priority is shared for all receivers of
        a given subscription.";
    }
    leaf subscription-priority {
     type uint8;
         description
          "Relative priority for a subscription.   Allows an underlying
           transport layer perform informed load balance allocations
           between various subscriptions";
    }
    leaf subscription-dependency {
      type string;
        description
       "Provides the Subscription ID of a parent subscription
        without which this subscription should not exist. In
        other words, there is no reason to stream these objects
        if another subscription is missing.";
    }
  }

  rpc establish-subscription {
    description
      "This RPC allows a subscriber to establish a subscription
       on its own behalf.  If successful, the subscription
       remains in effect for the duration of the subscriber’s
       association with the publisher, or until the subscription
       is terminated by virtue of a delete-subscription request.
       In case an error (as indicated by subscription-result)
       is returned, the subscription is
       not establishd.  In that case, the RPC output
       MAY include suggested parameter settings
       that would have a high likelihood of succeeding in a
       subsequent establish-subscription request.";
    input {
      uses subscription-info;
      uses update-policy;
      uses subscription-qos;
    }
    output {
      leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
          "Indicates whether subscription is operational,
           or if a problem was encountered.";
      }
      choice result {
        description
          "Depending on the subscription result, different
           data is returned.";
        case success {
          description
            "This case is used when the subscription request
             was successful and a subscription was established as
             a result";
          leaf subscription-id {
            type subscription-id;
            mandatory true;
            description
              "Identifier used for this subscription.";
          }
        }
        case no-success {
          description
            "This case applies when a subscription request
             was not successful and no subscription was
             established as a result.  In this case,
             information MAY be returned that indicates
             suggested parameter settings that would have a
             high likelihood of succeeding in a subsequent
             establish-subscription request.";
          uses subscription-info;
          uses update-policy;
          uses subscription-qos;
        }
      }
    }
  }
  rpc modify-subscription {
    description
      "This RPC allows a subscriber to modify a subscription
       that was previously established using establish-subscription.
       If successful, the subscription
       remains in effect for the duration of the subscriber’s
       association with the publisher, or until the subscription
       is terminated by virtue of a delete-subscription request.
       In case an error is returned (as indicated by
       subscription-result), the subscription is
       not modified and the original subscription parameters
       remain in effect.  In that case, the rpc error response
       MAY include suggested parameter settings
       that would have a high likelihood of succeeding in a
       subsequent modify-subscription request.";
    input {
      leaf subscription-id {
        type subscription-id;
        description
          "Identifier to use for this subscription.";
        }
        uses subscription-info;
        uses update-policy;
        uses subscription-qos;
    }
    output {
      leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
          "Indicates whether subscription was modified
           or if a problem was encountered.
           In case the subscription-result has a value
           other than OK, the original subscription was not
           changed.";
      }
      uses subscription-info;
      uses update-policy;
      uses subscription-qos;
    }
  }
  rpc delete-subscription {
    description
      "This RPC allows a subscriber to delete a subscription that
       was previously established using establish-subscription.";
    input {
      leaf subscription-id {
        type subscription-id;
        description
          "Identifier of the subscription that is to be deleted.
           Only subscriptions that were established using
           establish-subscription can be deleted via this RPC.";
      }
    }
  }
  notification push-update {
    description
      "This notification contains a periodic push update.
       This notification shall only be sent to receivers
       of a subscription; it does not constitute a general-purpose
       notification.";
    leaf subscription-id {
      type subscription-id;
      mandatory true;
      description
        "This references the subscription because of which the
         notification is sent.";
    }
    leaf time-of-update {
      type yang:date-and-time;
      description
        "This leaf contains the time of the update.";
    }
    choice encoding {
      description
        "Distinguish between the proper encoding that was specified
         for the subscription";
      case encode-xml {
        description
          "XML encoding";
        leaf datastore-contents-xml {
          type datastore-contents-xml;
          description
            "This contains data encoded in XML,
             per the subscription.";
        }
      }
      case encode-json {
        if-feature "json";
        description
          "JSON encoding";
        leaf datastore-contents-json {
          type datastore-contents-json;
          description
            "This leaf contains data encoded in JSON,
             per the subscription.";
        }
      }
    }
  }
  notification push-change-update {
    if-feature "on-change";
    description
      "This notification contains an on-change push update.
       This notification shall only be sent to the receivers
       of a subscription; it does not constitute a general-purpose
       notification.";
    leaf subscription-id {
      type subscription-id;
      mandatory true;
      description
        "This references the subscription because of which the
         notification is sent.";
    }
    leaf time-of-update {
      type yang:date-and-time;
      description
```

```
          "This leaf contains the time of the update, i.e. the
           time at which the change was observed.";
      }
      choice encoding {
        description
          "Distinguish between the proper encoding that was specified
           for the subscription";
        case encode-xml {
          description
            "XML encoding";
          leaf datastore-changes-xml {
            type datastore-changes-xml;
            description
              "This contains datastore contents that has changed
               since the previous update, per the terms of the
               subscription.  Changes are encoded analogous to
               the syntax of a corresponding Netconf edit-config
               operation.";
          }
        }
        case encode-json {
          if-feature "json";
          description
            "JSON encoding";
          leaf datastore-changes-yang {
            type datastore-changes-json;
            description
              "This contains datastore contents that has changed
               since the previous update, per the terms of the
               subscription.  Changes are encoded analogous
               to the syntax of a corresponding RESTCONF yang-patch
               operation.";
          }
        }
      }
    }
    notification subscription-started {
      description
        "This notification indicates that a subscription has
         started and data updates are beginning to be sent.
         This notification shall only be sent to receivers
         of a subscription; it does not constitute a general-purpose
         notification.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      uses subscription-info;
      uses update-policy;
      uses subscription-qos;
    }
    notification subscription-suspended {
      description
        "This notification indicates that a suspension of the
         subscription by the server has occurred.  No further
         datastore updates will be sent until subscription
         resumes.
         This notification shall only be sent to receivers
         of a subscription; it does not constitute a general-purpose
         notification.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      leaf reason {
        type subscription-susp-reason;
        description
          "Provides a reason for why the subscription was
           suspended.";
      }
    }
    notification subscription-resumed {
      description
        "This notification indicates that a subscription that had
         previously been suspended has resumed. Datastore updates
         will once again be sent.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
    }
    notification subscription-modified {
      description
        "This notification indicates that a subscription has
         been modified.  Datastore updates sent from this point
         on will conform to the modified terms of the
         subscription.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      uses subscription-info;
      uses update-policy;
      uses subscription-qos;
    }
    notification subscription-terminated {
      description
        "This notification indicates that a subscription has been
         terminated.";
      leaf subscription-id {
        type subscription-id;
        mandatory true;
        description
          "This references the affected subscription.";
      }
      leaf reason {
        type subscription-term-reason;
        description
          "Provides a reason for why the subscription was
           terminated.";
      }
    }
    container update-streams {
      config false;
      description
        "This container contains a leaf list of built-in
         streams that are provided by the system.";
      leaf-list update-stream {
        type update-stream;
        description
          "Identifies a built-in stream that is supported by the
           system.  Streams are associated with their own identities,
           each of which carries a special semantics.";
      }
    }
    container filters {
      description
        "This container contains a list of configurable filters
         that can be applied to subscriptions.  This facilitates
         the reuse of complex filters once defined.";
      list filter {
        key "filter-id";
        description
          "A list of configurable filters that can be applied to
           subscriptions.";
        leaf filter-id {
          type filter-id;
          description
            "An identifier to differentiate between filters.";
        }
        uses datatree-filter;
      }
    }
    container subscription-config {
      if-feature "configured-subscriptions";
      description
        "Contains the list of subscriptions that are configured,
         as opposed to established via RPC or other means.";
      list yang-push-subscription {
        key "subscription-id";
        description
          "Content of a yang-push subscription.";
        leaf subscription-id {
          type subscription-id;
          description
            "Identifier to use for this subscription.";
        }
        uses subscription-info;
        uses update-policy;
        uses receiver-info;
        uses push-source-info;
        uses subscription-qos;
      }
    }
    container subscriptions {
      config false;
      description
        "Contains the list of currently active subscriptions,
         i.e. subscriptions that are currently in effect,
         used for subscription management and monitoring purposes.
         This includes subscriptions that have been setup via RPC
         primitives, e.g. establish-subscription, delete-subscription,
         and modify-subscription, as well as subscriptions that
         have been established via configuration.";
      list yang-push-subscription {
        key "subscription-id";
        config false;
        description
          "Content of a yang-push subscription.
           Subscriptions can be established using a control channel
           or RPC, or be established through configuration.";
        leaf subscription-id {
          type subscription-id;
          description
```

```
          "Identifier of this subscription.";
      }
      leaf configured-subscription {
        if-feature "configured-subscriptions";
        type empty;
        description
          "The presence of this leaf indicates that the
           subscription originated from configuration, not through
           a control channel or RPC.";
      }
      leaf subscription-status {
        type identityref {
          base subscription-stream-status;
        }
        description
          "The status of the subscription.";
      }
      uses subscription-info;
      uses update-policy;
      uses receiver-info;
      uses push-source-info;
      uses subscription-qos;
    }
  }
}

<CODE ENDS>
```

6. Security Considerations

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

A subscription could be used to retrieve data in subtrees that a client has not authorized access to. Therefore it is important that data pushed based on subscriptions is authorized in the same way that regular data retrieval operations are. Data being pushed to a client needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

A subscription could be configured on another receiver’s behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver which doesn’t even support subscriptions. Clients which do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, the Netconf Authorization Control Model SHOULD be used to control and restrict authorization of subscription configuration.

7. References

7.1. Normative References

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, DOI 10.17487/RFC1157, May 1990, <http://www.rfc-editor.org/info/rfc1157>.

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

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

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

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 5277, February 2012.

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

7.2. Informative References

[I-D.clemm-netmod-mount]
Clemm, A., Medved, J., and E. Voit, "Mounting YANG-defined information from remote datastores", draft-clemm-netmod-mount-03 (work in progress), April 2015.

[I-D.gonzalez-netconf-5277bis]
Gonzalez Prieto, A., Clemm, A., Voit, E., Tripathy, A., and E. Nilsen-Nygaard, "Mounting YANG-defined information from remote datastores", draft-clemm-netmod-mount-03 (work in progress), March 2016.

[I-D.i2rs-pub-sub-requirements]
Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-05 (work in progress), February 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-10, March 2016.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-08 (work in progress), March 2016.

[I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-07 (work in progress), January 2016.

[I-D.voit-netmod-yang-mount-requirements]
Voit, E., Clemm, A., and S. Mertens, "Requirements for Peer Mounting of YANG subtrees from Remote Datastores", draft-voit-netmod-yang-mount-requirements-00 (work in progress), March 2016.

Authors’ Addresses

Alexander Clemm
Cisco Systems

EMail: alex@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

EMail: albertgo@cisco.com

Eric Voit
Cisco Systems

EMail: evoit@cisco.com

Ambika Prasad Tripathy
Cisco Systems

EMail: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

EMail: einarnn@cisco.com

NETCONF Working Group   K. Watsen
Internet-Draft   Juniper Networks
Intended status: Standards Track   M. Abrahamsson
Expires: July 19, 2019   T-Systems
                                                               I. Farrer
                                                     Deutsche Telekom AG
                                                        January 15, 2019

Secure Zero Touch Provisioning (SZTP)
draft-ietf-netconf-zerotouch-29

Abstract

This draft presents a technique to securely provision a networking device when it is booting in a factory-default state. Variations in the solution enables it to be used on both public and private networks. The provisioning steps are able to update the boot image, commit an initial configuration, and execute arbitrary scripts to address auxiliary needs. The updated device is subsequently able to establish secure connections with other systems. For instance, a device may establish NETCONF (RFC 6241) and/or RESTCONF (RFC 8040) connections with deployment-specific network management systems.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in the IANA Considerations section contains placeholder values for DHCP options pending IANA assignment. Please apply the following replacements:

- "TBD1" --> the assigned value for id-ct-sztpConveyedInfoXML

- "TBD2" --> the assigned value for id-ct-sztpConveyedInfoJSON

- "TBD_IANA_URL" --> the assigned URL for the IANA registry

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- "XXXX" --> the assigned numerical RFC value for this draft

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- "2019-01-15" --> the publication date of this draft

The following one Appendix section is to be removed prior to publication:

- Appendix D. Change Log

Status of This Memo

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

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

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

This Internet-Draft will expire on July 19, 2019.

Copyright Notice

Table of Contents

1. Introduction

A fundamental business requirement for any network operator is to reduce costs where possible. For network operators, deploying devices to many locations can be a significant cost, as sending trained specialists to each site for installations is both cost prohibitive and does not scale.

This document defines Secure Zero Touch Provisioning (SZTP), a bootstrapping strategy enabling devices to securely obtain bootstrapping data with no installer action beyond physical placement and connecting network and power cables. As such, SZTP enables non-technical personnel to bring up devices in remote locations without the need for any operator input.

The SZTP solution includes updating the boot image, committing an initial configuration, and executing arbitrary scripts to address auxiliary needs. The updated device is subsequently able to establish secure connections with other systems. For instance, a devices may establish NETCONF [RFC8040] and/or RESTCONF [RFC6241] connections with deployment-specific network management systems.

This document primarily regards physical devices, where the setting of the device's initial state, described in Section 5.1, occurs during the device's manufacturing process. The SZTP solution may be extended to support virtual machines or other such logical constructs, but details for how this can be accomplished is left for future work.

1.1. Use Cases

- Device connecting to a remotely administered network

  This use-case involves scenarios, such as a remote branch office or convenience store, whereby a device connects as an access gateway to an ISP's network. Assuming it is not possible to customize the ISP's network to provide any bootstrapping support, and with no other nearby device to leverage, the device has no recourse but to reach out to an Internet-based bootstrap server to bootstrap from.

- Device connecting to a locally administered network

  This use-case covers all other scenarios and differs only in that the device may additionally leverage nearby devices, which may direct it to use a local service to bootstrap from. If no such information is available, or the device is unable to use the information provided, it can then reach out to the network just as it would for the remotely administered network use-case.

Conceptual workflows for how SZTP might be deployed are provided in Appendix C.

1.2. Terminology

This document uses the following terms (sorted by name):

Artifact: The term "artifact" is used throughout to represent any of the three artifacts defined in Section 3 (conveyed information, ownership voucher, and owner certificate). These artifacts collectively provide all the bootstrapping data a device may use.

Bootstrapping Data: The term "bootstrapping data" is used throughout this document to refer to the collection of data that a device may obtain during the bootstrapping process. Specifically, it refers to the three artifacts conveyed information, owner certificate, and ownership voucher, as described in Section 3.

Bootstrap Server: The term "bootstrap server" is used within this document to mean any RESTCONF server implementing the YANG module defined in Section 7.3.

Conveyed Information: The term "conveyed information" is used herein to refer either redirect information or onboarding information. Conveyed information is one of the three bootstrapping artifacts described in Section 3.

Device: The term "device" is used throughout this document to refer to a network element that needs to be bootstrapped. See Section 5 for more information about devices.

Manufacturer: The term "manufacturer" is used herein to refer to the manufacturer of a device or a delegate of the manufacturer.

Network Management System (NMS): The acronym "NMS" is used throughout this document to refer to the deployment-specific management system that the bootstrapping process is responsible for introducing devices to. From a device’s perspective, when the bootstrapping process has completed, the NMS is a NETCONF or RESTCONF client.

Onboarding Information: The term "onboarding information" is used herein to refer to one of the two types of "conveyed information" defined in this document, the other being "redirect information". Onboarding information is formally defined by the "onboarding-information" YANG-data structure in Section 6.3.

Onboarding Server: The term "onboarding server" is used herein to refer to a bootstrap server that only returns onboarding information.

Owner: The term "owner" is used throughout this document to refer to the person or organization that purchased or otherwise owns a device.

Owner Certificate: The term "owner certificate" is used in this document to represent an X.509 certificate that binds an owner identity to a public key, which a device can use to validate a signature over the conveyed information artifact. The owner certificate may be communicated along with its chain of intermediate certificates leading up to a known trust anchor. The owner certificate is one of the three bootstrapping artifacts described in Section 3.

Ownership Voucher: The term "ownership voucher" is used in this document to represent the voucher artifact defined in [RFC8366]. The ownership voucher is used to assign a device to an owner. The ownership voucher is one of the three bootstrapping artifacts described in Section 3.

Redirect Information: The term "redirect information" is used herein to refer to one of the two types of "conveyed information" defined in this document, the other being "onboarding information". Redirect information is formally defined by the "redirect-information" YANG-data structure in Section 6.3.

Redirect Server: The term "redirect server" is used to refer to a bootstrap server that only returns redirect information. A redirect server is particularly useful when hosted by a manufacturer, as a well-known (e.g., Internet-based) resource to redirect devices to deployment-specific bootstrap servers.

Signed Data: The term "signed data" is used throughout to mean conveyed information that has been signed, specifically by a private key possessed by a device’s owner.

Unsigned Data: The term "unsigned data" is used throughout to mean conveyed information that has not been signed.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.4. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

2. Types of Conveyed Information

This document defines two types of conveyed information that devices can access during the bootstrapping process. These conveyed information types are described in this section. Examples are provided in Section 6.2

2.1. Redirect Information

Redirect information redirects a device to another bootstrap server. Redirect information encodes a list of bootstrap servers, each specifying the bootstrap server’s hostname (or IP address), an optional port, and an optional trust anchor certificate that the device can use to authenticate the bootstrap server with.

Redirect information is YANG modeled data formally defined by the "redirect-information" container in the YANG module presented in Section 6.3. This container has the tree diagram shown below.

```
            +--:(redirect-information)
               +-- redirect-information
                  +-- bootstrap-server* [address]
                     +-- address         inet:host
                     +-- port?           inet:port-number
                     +-- trust-anchor?   cms
```

Redirect information may be trusted or untrusted. The redirect information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device’s owner. In all other cases, the redirect information is untrusted.

Trusted redirect information is useful for enabling a device to establish a secure connection to a specified bootstrap server, which is possible when the redirect information includes the bootstrap server's trust anchor certificate.

Untrusted redirect information is useful for directing a device to a bootstrap server where signed data has been staged for it to obtain. Note that, when the redirect information is untrusted, devices discard any potentially included trust anchor certificates.

How devices process redirect information is described in Section 5.5.

2.2. Onboarding Information

Onboarding information provides data necessary for a device to bootstrap itself and establish secure connections with other systems. As defined in this document, onboarding information can specify details about the boot image a device must be running, specify an initial configuration the device must commit, and specify scripts that the device must successfully execute.

Onboarding information is YANG modeled data formally defined by the "onboarding-information" container in the YANG module presented in Section 6.3. This container has the tree diagram shown below.

```
         +--:(onboarding-information)
            +-- onboarding-information
               +-- boot-image
               |  +-- os-name?              string
               |  +-- os-version?           string
               |  +-- download-uri*         inet:uri
               |  +-- image-verification* [hash-algorithm]
               |     +-- hash-algorithm    identityref
               |     +-- hash-value        yang:hex-string
               +-- configuration-handling?      enumeration
               +-- pre-configuration-script?    script
               +-- configuration?               binary
               +-- post-configuration-script?   script
```

Onboarding information must be trusted for it to be of any use to a device. There is no option for a device to process untrusted onboarding information.

Onboarding information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's owner. In all other cases, the onboarding information is untrusted.

How devices process onboarding information is described in Section 5.6.

3. Artifacts

This document defines three artifacts that can be made available to devices while they are bootstrapping. Each source of bootstrapping data specifies how it provides the artifacts defined in this section (see Section 4).

3.1. Conveyed Information

The conveyed information artifact encodes the essential bootstrapping data for the device. This artifact is used to encode the redirect information and onboarding information types discussed in Section 2.

The conveyed information artifact is a CMS structure, as described in [RFC5652], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690 [ITU.X690.2015]. The CMS structure MUST contain content conforming to the YANG module specified in Section 6.3.

The conveyed information CMS structure may encode signed or unsigned bootstrapping data. When the bootstrapping data is signed, it may also be encrypted but, from a terminology perspective, it is still "signed data" Section 1.2.

When the conveyed information artifact is unsigned, as it might be when communicated over trusted channels, the CMS structure's top-most content type MUST be one of the OIDs described in Section 10.3 (i.e., id-ct-sztpConveyedInfoXML or id-ct-sztpConveyedInfoJSON), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing "conveyed-information" data in the expected encoding.

When the conveyed information artifact is unsigned and encrypted, as it might be when communicated over trusted channels but, for some reason, the operator wants to ensure that only the device is able to see the contents, the CMS structure's top-most content type MUST be the OID id-envelopedData (1.2.840.113549.1.7.3). Furthermore, the encryptedContentInfo's content type MUST be one of the OIDs described in Section 10.3 (i.e., id-ct-sztpConveyedInfoXML or id-ct-sztpConveyedInfoJSON), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing "conveyed-information" data in the expected encoding.

When the conveyed information artifact is signed, as it might be when communicated over untrusted channels, the CMS structure's top-most content type MUST be the OID id-signedData (1.2.840.113549.1.7.2). Furthermore, the inner eContentType MUST be one of the OIDs described in Section 10.3 (i.e., id-ct-sztpConveyedInfoXML or id-ct-sztpConveyedInfoJSON), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content or eContent is an octet string containing "conveyed-information" data in the expected encoding.

When the conveyed information artifact is signed and encrypted, as it might be when communicated over untrusted channels and privacy is important, the CMS structure's top-most content type MUST be the OID id-envelopedData (1.2.840.113549.1.7.3). Furthermore, the encryptedContentInfo's content type MUST be the OID id-signedData (1.2.840.113549.1.7.2), whose eContentType MUST be one of the OIDs described in Section 10.3 (i.e., id-ct-sztpConveyedInfoXML or id-ct-sztpConveyedInfoJSON), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content or eContent is an octet string containing "conveyed-information" data in the expected encoding.

3.2. Owner Certificate

The owner certificate artifact is an X.509 certificate [RFC5280] that is used to identify an "owner" (e.g., an organization). The owner certificate can be signed by any certificate authority (CA). The owner certificate either MUST have no Key Usage specified or the Key Usage MUST at least set the "digitalSignature" bit. The values for the owner certificate's "subject" and/or "subjectAltName" are not constrained by this document.

The owner certificate is used by a device to verify the signature over the conveyed information artifact (Section 3.1) that the device should have also received, as described in Section 3.5. In particular, the device verifies the signature using the public key in the owner certificate over the content contained within the conveyed information artifact.

The owner certificate artifact is formally a CMS structure, as specified by [RFC5652], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690 [ITU.X690.2015].

The owner certificate CMS structure MUST contain the owner certificate itself, as well as all intermediate certificates leading to the "pinned-domain-cert" certificate specified in the ownership voucher. The owner certificate artifact MAY optionally include the "pinned-domain-cert" as well.

In order to support devices deployed on private networks, the owner certificate CMS structure MAY also contain suitably fresh, as determined by local policy, revocation objects (e.g., CRLs). Having these revocation objects stapled to the owner certificate may obviate the need for the device to have to download them dynamically using the CRL distribution point or an OCSP responder specified in the associated certificates.

When unencrypted, the owner certificate artifact's CMS structure's top-most content type MUST be the OID id-signedData (1.2.840.113549.1.7.2). The inner SignedData structure is the degenerate form, whereby there are no signers, that is commonly used to disseminate certificates and revocation objects.

When encrypted, the owner certificate artifact's CMS structure's top-most content type MUST be the OID id-envelopedData (1.2.840.113549.1.7.3), and the encryptedContentInfo's content type MUST be the OID id-signedData (1.2.840.113549.1.7.2), whereby the inner SignedData structure is the degenerate form that has no signers commonly used to disseminate certificates and revocation objects.

3.3. Ownership Voucher

The ownership voucher artifact is used to securely identify a device's owner, as it is known to the manufacturer. The ownership voucher is signed by the device's manufacturer.

The ownership voucher is used to verify the owner certificate (Section 3.2) that the device should have also received, as described in Section 3.5. In particular, the device verifies that the owner certificate has a chain of trust leading to the trusted certificate included in the ownership voucher ("pinned-domain-cert"). Note that this relationship holds even when the owner certificate is a self-signed certificate, and hence also the pinned-domain-cert.

When unencrypted, the ownership voucher artifact is as defined in [RFC8366]. As described, it is a CMS structure whose top-most content type MUST be the OID id-signedData (1.2.840.113549.1.7.2), whose eContentType MUST be OID id-ct-animaJSONVoucher (1.2.840.113549.1.9.16.1), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing ietf-voucher data in the expected encoding.

When encrypted, the ownership voucher artifact's CMS structure's top-most content type MUST be the OID id-envelopedData (1.2.840.113549.1.7.3), and the encryptedContentInfo's content type MUST be the OID id-signedData (1.2.840.113549.1.7.2), whose eContentType MUST be OID id-ct-animaJSONVoucher (1.2.840.113549.1.9.16.1), or the OID id-data (1.2.840.113549.1.7.1). When the OID id-data is used, the encoding (JSON, XML, etc.) SHOULD be communicated externally. In either case, the associated content is an octet string containing ietf-voucher data in the expected encoding.

3.4. Artifact Encryption

Each of the three artifacts MAY be individually encrypted. Encryption may be important in some environments where the content is considered sensitive.

Each of the three artifacts are encrypted in the same way, by the unencrypted form being encapsulated inside a CMS EnvelopedData type.

As a consequence, both the conveyed information and ownership voucher artifacts are signed and then encrypted, never encrypted and then signed.

This sequencing has the advantage of shrouding the signer's certificate, and ensuring that the owner knows the content being signed. This sequencing further enables the owner to inspect an unencrypted voucher obtained from a manufacturer and then encrypt the voucher later themselves, perhaps while also stapling in current revocation objects, when ready to place the artifact in an unsafe location.

When encrypted, the CMS MUST be encrypted using a secure device identity certificate for the device. This certificate MAY be the same as the TLS-level client certificate the device uses when connecting to bootstrap servers. The owner must possess the device's identity certificate at the time of encrypting the data. How the owner comes to posses the device's identity certificate for this purpose is outside the scope of this document.

3.5. Artifact Groupings

The previous sections discussed the bootstrapping artifacts, but only certain groupings of these artifacts make sense to return in the various bootstrapping situations described in this document. These groupings are:

Unsigned Data: This artifact grouping is useful for cases when transport level security can be used to convey trust (e.g., HTTPS), or when the conveyed information can be processed in a provisional manner (i.e. unsigned redirect information).

Signed Data, without revocations: This artifact grouping is useful when signed data is needed (i.e., because the data is obtained from an untrusted source and it cannot be processed provisionally) and either revocations are not needed or the revocations can be obtained dynamically.

Signed Data, with revocations: This artifact grouping is useful when signed data is needed (i.e., because the data is obtained from an untrusted source and it cannot be processed provisionally), and revocations are needed, and the revocations cannot be obtained dynamically.

The presence of each artifact, and any distinguishing characteristics, are identified for each artifact grouping in the table below ("yes/no" regards if the artifact is present in the artifact grouping):

Artifact Grouping	Conveyed Information	Ownership Voucher	Owner Certificate
Unsigned Data	Yes, no sig	No	No
Signed Data, without revocations	Yes, with sig	Yes, without revocations	Yes, without revocations
Signed Data, with revocations	Yes, with sig	Yes, with revocations	Yes, with revocations

4. Sources of Bootstrapping Data

This section defines some sources for bootstrapping data that a device can access. The list of sources defined here is not meant to be exhaustive. It is left to future documents to define additional sources for obtaining bootstrapping data.

For each source of bootstrapping data defined in this section, details are given for how the three artifacts listed in Section 3 are provided.

4.1. Removable Storage

A directly attached removable storage device (e.g., a USB flash drive) MAY be used as a source of SZTP bootstrapping data.

Use of a removable storage device is compelling, as it does not require any external infrastructure to work. It is notable that the raw boot image file can also be located on the removable storage device, enabling a removable storage device to be a fully self-standing bootstrapping solution.

To use a removable storage device as a source of bootstrapping data, a device need only detect if the removable storage device is plugged in and mount its filesystem.

A removable storage device is an untrusted source of bootstrapping data. This means that the information stored on the removable storage device either MUST be signed or MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a removable storage device presents itself as a filesystem, the bootstrapping artifacts need to be presented as files. The three artifacts defined in Section 3 are mapped to files below.

Artifact to File Mapping:

Conveyed Information: Mapped to a file containing the binary artifact described in Section 3.1 (e.g., conveyed-information.cms).

Owner Certificate: Mapped to a file containing the binary artifact described in Section 3.2 (e.g., owner-certificate.cms).

Ownership Voucher: Mapped to a file containing the binary artifact described in Section 3.3 (e.g., ownership-voucher.cms or ownership-voucher.vcj).

The format of the removable storage device’s filesystem and the naming of the files are outside the scope of this document. However, in order to facilitate interoperability, it is RECOMMENDED devices support open and/or standards based filesystems. It is also RECOMMENDED that devices assume a file naming convention that enables more than one instance of bootstrapping data (i.e., for different devices) to exist on a removable storage device. The file naming convention SHOULD additionally be unique to the manufacturer, in order to enable bootstrapping data from multiple manufacturers to exist on a removable storage device.

4.2. DNS Server

A DNS server MAY be used as a source of SZTP bootstrapping data.

Using a DNS server may be a compelling option for deployments having existing DNS infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

DNS is an untrusted source of bootstrapping data. Even if DNSSEC [RFC6698] is used to authenticate the various DNS resource records (e.g., A, AAAA, CERT, TXT, and TLSA), the device cannot be sure that the domain returned to it from e.g., a DHCP server, belongs to its rightful owner. This means that the information stored in the DNS records either MUST be signed (per this document, not DNSSEC), or MUST be information that can be processed provisionally (e.g., unsigned redirect information).

4.2.1. DNS Queries

Devices claiming to support DNS as a source of bootstrapping data MUST first query for device-specific DNS records and, only if doing so does not result in a successful bootstrap, then MUST query for device-independent DNS records.

For each of the device-specific and device-independent queries, devices MUST first query using multicast DNS [RFC6762] and, only if doing so does not result in a successful bootstrap, then MUST query again using unicast DNS [RFC1035] [RFC7766], assuming the address of a DNS server is known, such as it may be using techniques similar to those described in Section 11 of [RFC6763], which is referenced a few times in this document, even though this document does not itself use DNS-SD (RFC 6763 is identified herein as an Informative reference).

When querying for device-specific DNS records, devices MUST query for TXT records [RFC1035] under "<serial-number>._sztp", where <serial-number> is the device’s serial number (the same value as in the device’s secure device identity certificate), and "_sztp" is the globally scoped DNS attribute registered by this document in Section 10.7.

Example device-specific DNS record queries:

```
   TXT in <serial-number>._sztp.local.  (multicast)
   TXT in <serial-number>._sztp.<domain>.  (unicast)
```

When querying for device-independent DNS records, devices MUST query for SRV records [RFC2782] under "_sztp._tcp", where "_sztp" is the service name registered by this document in Section 10.6, and "_tcp" is the globally scoped DNS attribute registered by [I-D.ietf-dnsop-attrleaf].

Note that a device-independent response is anyway only able to encode unsigned data, since signed data necessitates the use of a device-specific ownership voucher. Use of SRV records maximumly leverages existing DNS standards. A response containing multiple SRV records is comparable to an unsigned redirect information's list of bootstrap servers.

Example device-independent DNS record queries:

```
SRV in _sztp._tcp.local.  (multicast)
SRV in _sztp._tcp.<domain>.  (unicast)
```

4.2.2. DNS Response for Device-Specific Queries

For device-specific queries, the three bootstrapping artifacts defined in Section 3 are encoded into the TXT records using key/value pairs, similar to the technique described in Section 6.3 of [RFC6763].

Artifact to TXT Record Mapping:

Conveyed Information: Mapped to a TXT record having the key "ci" and the value being the binary artifact described in Section 3.1.

Owner Certificate: Mapped to a TXT record having the key "oc" and the value being the binary artifact described in Section 3.2.

Ownership Voucher: Mapped to a TXT record having the key "ov" and the value being the binary artifact described in Section 3.3.

Devices MUST ignore any other keys that may be returned.

Note that, despite the name, TXT records can and SHOULD (per Section 6.5 of [RFC6763]) encode binary data.

Following is an example of a device-specific response, as it might be presented by a user-agent, containing signed data. This example assumes that the device's serial number is "<serial-number>", the domain is "example.com", and that "<binary data>" represents the binary artifact:

```
     <serial-number>._sztp.example.com. 3600 IN TXT "ci=<binary data>"
     <serial-number>._sztp.example.com. 3600 IN TXT "oc=<binary data>"
     <serial-number>._sztp.example.com. 3600 IN TXT "ov=<binary data>"
```

Note that, in the case that "ci" encodes unsigned data, the "oc" and "ov" keys would not be present in the response.

4.2.3. DNS Response for Device-Independent Queries

For device-independent queries, the three bootstrapping artifacts defined in Section 3 are encoded into the SVR records as follows.

Artifact to SRV Record Mapping:

Conveyed Information: This artifact is not supported directly. Instead, the essence of unsigned redirect information is mapped to SVR records per [RFC2782].

Owner Certificate: Not supported. Device-independent responses are never encode signed data, and hence there is no need for an owner certificate artifact.

Ownership Voucher: Not supported. Device-independent responses are never encode signed data, and hence there is no need for an ownership voucher artifact.

Following is an example of a device-independent response, as it might be presented by a user-agent, containing (effectively) unsigned redirect information to four bootstrap servers. This example assumes that the domain is "example.com" and that there are four bootstrap servers "sztp[1-4]":

```
      _sztp._tcp.example.com. 1800 IN SRV 0 0 443 sztp1.example.com.
      _sztp._tcp.example.com. 1800 IN SRV 1 0 443 sztp2.example.com.
      _sztp._tcp.example.com. 1800 IN SRV 2 0 443 sztp3.example.com.
      _sztp._tcp.example.com. 1800 IN SRV 2 0 443 sztp4.example.com.
```

Note that, in this example, "sztp3" and "sztp4" have equal priority, and hence effectively represent a clustered pair of bootstrap servers. While "sztp1" and "sztp2" only have a single SRV record each, it may be that the record points to a load-balancer fronting a cluster of bootstrap servers.

While this document does not use DNS-SD [RFC6763], per Section 12.2 of that RFC, mDNS responses SHOULD also include all address records (type "A" and "AAAA") named in the SRV rdata.

4.2.4. Size of Signed Data

The signed data artifacts are large by DNS conventions. In the smallest-footprint scenario, they are each a few kilobytes in size. However, onboarding information can easily be several kilobytes in size, and has the potential to be many kilobytes in size.

All resource records, including TXT records, have an upper size limit of 65535 bytes, since "RDLENGTH" is a 16-bit field (Section 3.2.1 in [RFC1035]). If it is ever desired to encode onboarding information that exceeds this limit, the DNS records returned should instead encode redirect information, to direct the device to a bootstrap server from which the onboarding information can be obtained.

Given the expected size of the TXT records, it is unlikely that signed data will fit into a UDP-based DNS packet, even with the EDNS(0) Extensions [RFC6891] enabled. Depending on content, signed data may also not fit into a multicast DNS packet, which bounds the size to 9000 bytes, per Section 17 in [RFC6762]. Thus it is expected that DNS Transport over TCP [RFC7766] will be required in order to return signed data.

4.3. DHCP Server

A DHCP server MAY be used as a source of SZTP bootstrapping data.

Using a DHCP server may be a compelling option for deployments having existing DHCP infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

A DHCP server is an untrusted source of bootstrapping data. Thus the information stored on the DHCP server either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

However, unlike other sources of bootstrapping data described in this document, the DHCP protocol (especially DHCP for IPv4) is very limited in the amount of data that can be conveyed, to the extent that signed data cannot be communicated. This means that only unsigned redirect information can be conveyed via DHCP.

Since the redirect information is unsigned, it SHOULD NOT include the optional trust anchor certificate, as it takes up space in the DHCP message, and the device would have to discard it anyway. For this reason, the DHCP options defined in Section 8 do not enable the trust anchor certificate to be encoded.

From an artifact perspective, the three artifacts defined in Section 3 are mapped to the DHCP fields specified in Section 8 as follows.

Artifact to DHCP Option Fields Mapping:

Conveyed Information: This artifact is not supported directly. Instead, the essence of unsigned redirect information is mapped to the DHCP options described in Section 8.

Owner Certificate: Not supported. There is not enough space in the DHCP packet to hold an owner certificate artifact.

Ownership Voucher: Not supported. There is not enough space in the DHCP packet to hold an ownership voucher artifact.

4.4. Bootstrap Server

A bootstrap server MAY be used as a source of SZTP bootstrapping data. A bootstrap server is defined as a RESTCONF [RFC8040] server implementing the YANG module provided in Section 7.

Using a bootstrap server as a source of bootstrapping data is a compelling option as it MAY use transport-level security, obviating the need for signed data, which may be easier to deploy in some situations.

Unlike any other source of bootstrapping data described in this document, a bootstrap server is not only a source of data, but it can also receive data from devices using the YANG-defined "report-progress" RPC defined in the YANG module (Section 7.3). The "report-progress" RPC enables visibility into the bootstrapping process (e.g., warnings and errors), and provides potentially useful information upon completion (e.g., the device’s SSH host-keys).

A bootstrap server may be a trusted or an untrusted source of bootstrapping data, depending on if the device learned about the bootstrap server’s trust anchor from a trusted source. When a bootstrap server is trusted, the conveyed information returned from it MAY be signed. When the bootstrap server is untrusted, the conveyed information either MUST be signed or MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a bootstrap server presents data conforming to a YANG data model, the bootstrapping artifacts need to be mapped to YANG nodes. The three artifacts defined in Section 3 are mapped to "output" nodes of the "get-bootstrapping-data" RPC defined in Section 7.3 below.

Artifact to Bootstrap Server Mapping:

Conveyed Information: Mapped to the "conveyed-information" leaf in the output of the "get-bootstrapping-data" RPC.

Owner Certificate: Mapped to the "owner-certificate" leaf in the output of the "get-bootstrapping-data" RPC.

Ownership Voucher: Mapped to the "ownership-voucher" leaf in the output of the "get-bootstrapping-data" RPC.

SZTP bootstrap servers have only two endpoints, one for the "get-bootstrapping-data" RPC and one for the "report-progress" RPC. These RPCs use the authenticated RESTCONF username to isolate the execution of the RPC from other devices.

5. Device Details

Devices supporting the bootstrapping strategy described in this document MUST have the preconfigured state and bootstrapping logic described in the following sections.

5.1. Initial State

```
      +-------------------------------------------------------------+
      |                           <device>                          |
      |                                                             |
      | +---------------------------------------------------------+ |
      | |                   <read/write storage>                  | |
      | |                                                         | |
      | | 1. flag to enable SZTP bootstrapping set to "true"      | |
      | +---------------------------------------------------------+ |
      |                                                             |
      | +---------------------------------------------------------+ |
      | |                   <read-only storage>                   | |
      | |                                                         | |
      | | 2. TLS client cert & related intermediate certificates  | |
      | | 3. list of trusted well-known bootstrap servers         | |
      | | 4. list of trust anchor certs for bootstrap servers     | |
      | | 5. list of trust anchor certs for ownership vouchers    | |
      | +---------------------------------------------------------+ |
      |                                                             |
      |   +-----------------------------------------------------+   |
      |   |                 <secure storage>                    |   |
      |   |                                                     |   |
      |   |  6. private key for TLS client certificate          |   |
      |   |  7. private key for decrypting SZTP artifacts       |   |
      |   +-----------------------------------------------------+   |
      |                                                             |
      +-------------------------------------------------------------+
```

Each numbered item below corresponds to a numbered item in the diagram above.

1. Devices MUST have a configurable variable that is used to enable/ disable SZTP bootstrapping. This variable MUST be enabled by default in order for SZTP bootstrapping to run when the device first powers on. Because it is a goal that the configuration installed by the bootstrapping process disables SZTP bootstrapping, and because the configuration may be merged into the existing configuration, using a configuration node that relies on presence is NOT RECOMMENDED, as it cannot be removed by the merging process.

2. Devices that support loading bootstrapping data from bootstrap servers (see Section 4.4) SHOULD possess a TLS-level client certificate and any intermediate certificates leading to the certificate's well-known trust-anchor. The well-known trust anchor certificate may be an intermediate certificate or a self-signed root certificate. To support devices not having a client certificate, devices MAY, alternatively or in addition to, identify and authenticate themselves to the bootstrap server

using an HTTP authentication scheme, as allowed by Section 2.5 in [RFC8040]; however, this document does not define a mechanism for operator input enabling, for example, the entering of a password.

3. Devices that support loading bootstrapping data from well-known bootstrap servers MUST possess a list of the well-known bootstrap servers. Consistent with redirect information (Section 2.1, each bootstrap server can be identified by its hostname or IP address, and an optional port.

4. Devices that support loading bootstrapping data from well-known bootstrap servers MUST also possess a list of trust anchor certificates that can be used to authenticate the well-known bootstrap servers. For each trust anchor certificate, if it is not itself a self-signed root certificate, the device SHOULD also possess the chain of intermediate certificates leading up to and including the self-signed root certificate.

5. Devices that support loading signed data (see Section 1.2) MUST possess the trust anchor certificates for validating ownership vouchers. For each trust anchor certificate, if it is not itself a self-signed root certificate, the device SHOULD also possess the chain of intermediate certificates leading up to and including the self-signed root certificate.

6. Devices that support using a TLS-level client certificate to identify and authenticate themselves to a bootstrap server MUST possess the private key that corresponds to the public key encoded in the TLS-level client certificate. This private key SHOULD be securely stored, ideally in a cryptographic processor, such as a trusted platform module (TPM) chip.

7. Devices that support decrypting SZTP artifacts MUST posses the private key that corresponds to the public key encoded in the secure device identity certificate used when encrypting the artifacts. This private key SHOULD be securely stored, ideally in a cryptographic processor, such as a trusted platform module (TPM) chip. This private key MAY be the same as the one associated to the TLS-level client certificate used when connecting to bootstrap servers.

A YANG module representing this data is provided in Appendix A.

5.2. Boot Sequence

A device claiming to support the bootstrapping strategy defined in this document MUST support the boot sequence described in this section.

```
        Power On
            |
            v                           No
     1. SZTP bootstrapping configured ------> Boot normally
            |
            | Yes
            v
     2. For each supported source of bootstrapping data,
        try to load bootstrapping data from the source
            |
            |
            v                               Yes
     3. Able to bootstrap from any source? -----> Run with new config
            |
            | No
            v
     4. Loop back to Step 1.
```

Note: At any time, the device MAY be configured via an alternate provisioning mechanism (e.g., CLI).

Each numbered item below corresponds to a numbered item in the diagram above.

1. When the device powers on, it first checks to see if SZTP bootstrapping is configured, as is expected to be the case for the device’s preconfigured initial state. If SZTP bootstrapping is not configured, then the device boots normally.

2. The device iterates over its list of sources for bootstrapping data (Section 4). Details for how to processes a source of bootstrapping data are provided in Section 5.3.

3. If the device is able to bootstrap itself from any of the sources of bootstrapping data, it runs with the new bootstrapped configuration.

4. Otherwise the device MUST loop back through the list of bootstrapping sources again.

This document does not limit the simultaneous use of alternate provisioning mechanisms. Such mechanisms may include, for instance, a command line interface (CLI), a web-based user interface, or even another bootstrapping protocol. Regardless how it is configured, the configuration SHOULD unset the flag enabling SZTP bootstrapping discussed in Section 5.1.

5.3. Processing a Source of Bootstrapping Data

This section describes a recursive algorithm that devices can use to, ultimately, obtain onboarding information. The algorithm is recursive because sources of bootstrapping data may return redirect information, which causes the algorithm to run again, for the newly discovered sources of bootstrapping data. An expression that captures all possible successful sequences of bootstrapping data is: zero or more redirect information responses, followed by one onboarding information response.

An important aspect of the algorithm is knowing when data needs to be signed or not. The following figure provides a summary of options:

		Untrusted Source	Trusted Source
Kind of Bootstrapping Data		Can Provide?	Can Provide?
Unsigned Redirect Info	:	Yes+	Yes
Signed Redirect Info	:	Yes	Yes*
Unsigned Onboarding Info	:	No	Yes
Signed Onboarding Info	:	Yes	Yes*

The '+' above denotes that the source redirected to MUST return signed data, or more unsigned redirect information.

The '*' above denotes that, while possible, it is generally unnecessary for a trusted source to return signed data.

The recursive algorithm uses a conceptual global-scoped variable called "trust-state". The trust-state variable is initialized to FALSE. The ultimate goal of this algorithm is for the device to process onboarding information (Section 2.2) while the trust-state variable is TRUE.

If the source of bootstrapping data (Section 4) is a bootstrap server (Section 4.4), and the device is able to authenticate the bootstrap server using X.509 certificate path validation ([RFC6125], Section 6) to one of the device's preconfigured trust anchors, or to a trust anchor that it learned from a previous step, then the device MUST set trust-state to TRUE.

When establishing a connection to a bootstrap server, whether trusted or untrusted, the device MUST identify and authenticate itself to the bootstrap server using a TLS-level client certificate and/or an HTTP authentication scheme, per Section 2.5 in [RFC8040]. If both authentication mechanisms are used, they MUST both identify the same serial number.

When sending a client certificate, the device MUST also send all of the intermediate certificates leading up to, and optionally including, the client certificate’s well-known trust anchor certificate.

For any source of bootstrapping data (e.g., Section 4), if any artifact obtained is encrypted, the device MUST first decrypt it using the private key associated with the device certificate used to encrypt the artifact.

If the conveyed information artifact is signed, and the device is able to validate the signed data using the algorithm described in Section 5.4, then the device MUST set trust-state to TRUE; otherwise, if the device is unable to validate the signed data, the device MUST set trust-state to FALSE. Note, this is worded to cover the special case when signed data is returned even from a trusted source of bootstrapping data.

If the conveyed information artifact contains redirect information, the device MUST, within limits of how many recursive loops the device allows, process the redirect information as described in Section 5.5. Implementations MUST limit the maximum number of recursive redirects allowed; the maximum number of recursive redirects allowed SHOULD be no more than ten. This is the recursion step, it will cause the device to reenter this algorithm, but this time the data source will definitely be a bootstrap server, as redirect information is only able to redirect devices to bootstrap servers.

If the conveyed information artifact contains onboarding information, and trust-state is FALSE, the device MUST exit the recursive algorithm (as this is not allowed, see the figure above), returning to the bootstrapping sequence described in Section 5.2. Otherwise, the device MUST attempt to process the onboarding information as described in Section 5.6. Whether the processing of the onboarding information succeeds or fails, the device MUST exit the recursive algorithm, returning to the bootstrapping sequence described in Section 5.2, the only difference being in how it responds to the "Able to bootstrap from any source?" conditional described in the figure in the section.

5.4. Validating Signed Data

Whenever a device is presented signed data, it MUST validate the signed data as described in this section. This includes the case where the signed data is provided by a trusted source.

Whenever there is signed data, the device MUST also be provided an ownership voucher and an owner certificate. How all the needed artifacts are provided for each source of bootstrapping data is described in Section 4.

In order to validate signed data, the device MUST first authenticate the ownership voucher by validating its signature to one of its preconfigured trust anchors (see Section 5.1), which may entail using additional intermediate certificates attached to the ownership voucher. If the device has an accurate clock, it MUST verify that the ownership voucher was created in the past (i.e., "created-on" < now) and, if the "expires-on" leaf is present, the device MUST verify that the ownership voucher has not yet expired (i.e., now < "expires-on"). The device MUST verify that the ownership voucher's "assertion" value is acceptable (e.g., some devices may only accept the assertion value "verified"). The device MUST verify that the ownership voucher specifies the device's serial number in the "serial-number" leaf. If the "idevid-issuer" leaf is present, the device MUST verify that the value is set correctly. If the authentication of the ownership voucher is successful, the device extracts the "pinned-domain-cert" node, an X.509 certificate, that is needed to verify the owner certificate in the next step.

The device MUST next authenticate the owner certificate by performing X.509 certificate path verification to the trusted certificate extracted from the ownership voucher's "pinned-domain-cert" node. This verification may entail using additional intermediate certificates attached to the owner certificate artifact. If the ownership voucher's "domain-cert-revocation-checks" node's value is set to "true", the device MUST verify the revocation status of the certificate chain used to sign the owner certificate and, if suitably-fresh revocation status is unattainable or if it is determined that a certificate has been revoked, the device MUST NOT validate the owner certificate.

Finally, the device MUST verify that the conveyed information artifact was signed by the validated owner certificate.

If any of these steps fail, the device MUST invalidate the signed data and not perform any subsequent steps.

5.5. Processing Redirect Information

In order to process redirect information (Section 2.1), the device MUST follow the steps presented in this section.

Processing redirect information is straightforward; the device sequentially steps through the list of provided bootstrap servers until it can find one it can bootstrap from.

If a hostname is provided, and the hostname's DNS resolution is to more than one IP address, the device MUST attempt to connect to all of the DNS resolved addresses at least once, before moving on to the next bootstrap server. If the device is able to obtain bootstrapping data from any of the DNS resolved addresses, it MUST immediately process that data, without attempting to connect to any of the other DNS resolved addresses.

If the redirect information is trusted (e.g., trust-state is TRUE), and the bootstrap server entry contains a trust anchor certificate, then the device MUST authenticate the specified bootstrap server's TLS server certificate using X.509 certificate path validation ([RFC6125], Section 6) to the specified trust anchor. If the bootstrap server entry does not contain a trust anchor certificate device, the device MUST establish a provisional connection to the bootstrap server (i.e., by blindly accepting its server certificate), and set trust-state to FALSE.

If the redirect information is untrusted (e.g., trust-state is FALSE), the device MUST discard any trust anchors provided by the redirect information and establish a provisional connection to the bootstrap server (i.e., by blindly accepting its TLS server certificate).

5.6. Processing Onboarding Information

In order to process onboarding information (Section 2.2), the device MUST follow the steps presented in this section.

When processing onboarding information, the device MUST first process the boot image information (if any), then execute the pre-configuration script (if any), then commit the initial configuration (if any), and then execute the post-configuration script (if any), in that order.

When the onboarding information is obtained from a trusted bootstrap server, the device MUST send the "bootstrap-initiated" progress report, and send either a terminating "boot-image-installed-rebooting", "bootstrap-complete", or error specific progress report. If the bootstrap server's "get-bootstrapping-data" RPC-reply's "reporting-level" node is set to "verbose", the device MUST additionally send all appropriate non-terminating progress reports (e.g., initiated, warning, complete, etc.). Regardless of the reporting-level indicated by the bootstrap server, the device MAY send progress reports beyond the mandatory ones specified for the given reporting level.

When the onboarding information is obtained from an untrusted bootstrap server, the device MUST NOT send any progress reports to the bootstrap server, even though the onboarding information was, necessarily, signed and authenticated. Please be aware that bootstrap servers are recommended to promote untrusted connections to trusted connections, in the last paragraph of Section 9.6, so as to, in part, be able to collect progress reports from devices.

If the device encounters an error at any step, it MUST stop processing the onboarding information and return to the bootstrapping sequence described in Section 5.2. In the context of a recursive algorithm, the device MUST return to the enclosing loop, not back to the very beginning. Some state MAY be retained from the bootstrapping process (e.g., updated boot image, logs, remnants from a script, etc.). However, the retained state MUST NOT be active in any way (e.g., no new configuration or running of software), and MUST NOT hinder the ability for the device to continue the bootstrapping sequence (i.e., process onboarding information from another bootstrap server).

At this point, the specific ordered sequence of actions the device MUST perform is described.

If the onboarding information is obtained from a trusted bootstrap server, the device MUST send a "bootstrap-initiated" progress report. It is an error if the device does not receive back the "204 No Content" HTTP status line. If an error occurs, the device MUST try to send a "bootstrap-error" progress report before exiting.

The device MUST parse the provided onboarding information document, to extract values used in subsequent steps. Whether using a stream-based parser or not, if there is an error when parsing the onboarding information, and the device is connected to a trusted bootstrap server, the device MUST try to send a "parsing-error" progress report before exiting.

If boot image criteria are specified, the device MUST first determine if the boot image it is running satisfies the specified boot image criteria. If the device is already running the specified boot image, then it skips the remainder of this step. If the device is not running the specified boot image, then it MUST download, verify, and install, in that order, the specified boot image, and then reboot. If connected to a trusted bootstrap server, the device MAY try to send a "boot-image-mismatch" progress report. To download the boot image, the device MUST only use the URIs supplied by the onboarding information. To verify the boot image, the device MUST either use one of the verification fingerprints supplied by the onboarding information, or use a cryptographic signature embedded into the boot image itself using a mechanism not described by this document. Before rebooting, if connected to a trusted bootstrap server, the device MUST try to send a "boot-image-installed-rebooting" progress report. Upon rebooting, the bootstrapping process runs again, which will eventually come to this step again, but then the device will be running the specified boot image, and thus will move to processing the next step. If an error occurs at any step while the device is connected to a trusted bootstrap server (i.e., before the reboot), the device MUST try to send a "boot-image-error" progress report before exiting.

If a pre-configuration script has been specified, the device MUST execute the script, capture any output emitted from the script, and check if the script had any warnings or errors. If an error occurs while the device is connected to a trusted bootstrap server, the device MUST try to send a "pre-script-error" progress report before exiting.

If an initial configuration has been specified, the device MUST atomically commit the provided initial configuration, using the approach specified by the "configuration-handling" leaf. If an error occurs while the device is connected to a trusted bootstrap server, the device MUST try to send a "config-error" progress report before exiting.

If a post-configuration script has been specified, the device MUST execute the script, capture any output emitted from the script, and check if the script had any warnings or errors. If an error occurs while the device is connected to a trusted bootstrap server, the device MUST try to send a "post-script-error" progress report before exiting.

If the onboarding information was obtained from a trusted bootstrap server, and the result of the bootstrapping process did not disable the "flag to enable SZTP bootstrapping" described in Section 5.1, the device SHOULD send an "bootstrap-warning" progress report.

If the onboarding information was obtained from a trusted bootstrap server, the device MUST send a "bootstrap-complete" progress report. It is an error if the device does not receive back the "204 No Content" HTTP status line. If an error occurs, the device MUST try to send a "bootstrap-error" progress report before exiting.

At this point, the device has completely processed the bootstrapping data.

The device is now running its initial configuration. Notably, if NETCONF Call Home or RESTCONF Call Home [RFC8071] is configured, the device initiates trying to establish the call home connections at this time.

Implementation Notes:

Implementations may vary in how to ensure no unwanted state is retained when an error occurs.

Following are some guidelines for if the implementation chooses to undo previous steps:

* When an error occurs, the device must rollback the current step and any previous steps.

* Most steps are atomic. For example, the processing of a configuration is specified above as atomic, and the processing of scripts is similarly specified as atomic in the "ietf-sztp-conveyed-info" YANG module.

* In case the error occurs after the initial configuration was committed, the device must restore the configuration to the configuration that existed prior to the configuration being committed.

* In case the error occurs after a script had executed successfully, it may be helpful for the implementation to define scripts as being able to take a conceptual input parameter indicating that the script should remove its previously set state.

6. The Conveyed Information Data Model

This section defines a YANG 1.1 [RFC7950] module that is used to define the data model for the conveyed information artifact described in Section 3.1. This data model uses the "yang-data" extension statement defined in [RFC8040]. Examples illustrating this data model are provided in Section 6.2.

6.1. Data Model Overview

The following tree diagram provides an overview of the data model for the conveyed information artifact.

```
   module: ietf-sztp-conveyed-info

     yang-data conveyed-information:
       +-- (information-type)
          +--:(redirect-information)
          |  +-- redirect-information
          |     +-- bootstrap-server* [address]
          |        +-- address         inet:host
          |        +-- port?           inet:port-number
          |        +-- trust-anchor?   cms
          +--:(onboarding-information)
             +-- onboarding-information
                +-- boot-image
                |  +-- os-name?              string
                |  +-- os-version?           string
                |  +-- download-uri*         inet:uri
                |  +-- image-verification* [hash-algorithm]
                |     +-- hash-algorithm    identityref
                |     +-- hash-value        yang:hex-string
                +-- configuration-handling?      enumeration
                +-- pre-configuration-script?    script
                +-- configuration?               binary
                +-- post-configuration-script?   script
```

6.2. Example Usage

The following example illustrates how redirect information (Section 2.1) can be encoded using JSON.

```
   {
     "ietf-sztp-conveyed-info:redirect-information" : {
       "bootstrap-server" : [
         {
           "address" : "sztp1.example.com",
           "port" : 8443,
           "trust-anchor" : "base64encodedvalue=="
         },
         {
           "address" : "sztp2.example.com",
           "port" : 8443,
           "trust-anchor" : "base64encodedvalue=="
         },
         {
           "address" : "sztp3.example.com",
           "port" : 8443,
           "trust-anchor" : "base64encodedvalue=="
         }
       ]
     }
   }
```

The following example illustrates how onboarding information (Section 2.2) can be encoded using JSON.

[Note: '\' line wrapping for formatting only]

```
   {
     "ietf-sztp-conveyed-info:onboarding-information" : {
       "boot-image" : {
         "os-name" : "VendorOS",
         "os-version" : "17.2R1.6",
         "download-uri" : [ "http://some/path/to/raw/file" ],
         "image-verification" : [
           {
             "hash-algorithm" : "ietf-sztp-conveyed-info:sha-256",
             "hash-value" : "ba:ec:cf:a5:67:82:b4:10:77:c6:67:a6:22:ab:\
   7d:50:04:a7:8b:8f:0e:db:02:8b:f4:75:55:fb:c1:13:b2:33"
           }
         ]
       },
       "configuration-handling" : "merge",
       "pre-configuration-script" : "base64encodedvalue==",
       "configuration" : "base64encodedvalue==",
       "post-configuration-script" : "base64encodedvalue=="
     }
   }
```

6.3. YANG Module

The conveyed information data model is defined by the YANG module presented in this section.

This module uses data types defined in [RFC5280], [RFC5652], [RFC6234], and [RFC6991], an extension statement from [RFC8040], and an encoding defined in [ITU.X690.2015].

```
<CODE BEGINS> file "ietf-sztp-conveyed-info@2019-01-15.yang"
module ietf-sztp-conveyed-info {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-sztp-conveyed-info";
  prefix sztp-info;

  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Data Types";
  }
  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types";
  }
  import ietf-restconf {
    prefix rc;
    reference "RFC 8040: RESTCONF Protocol";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:   http://tools.ietf.org/wg/netconf
     WG List:  <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kwatsen@juniper.net>";

  description
   "This module defines the data model for the Conveyed
    Information artifact defined in RFC XXXX: Secure Zero Touch
    Provisioning (SZTP).

    The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
    'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
    'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
    are to be interpreted as described in BCP 14 (RFC 2119,
    RFC 8174) when, and only when, they appear in all
    capitals, as shown here.
```

```
         Copyright (c) 2019 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 (http://trustee.ietf.org/license-info)

         This version of this YANG module is part of RFC XXXX; see the
         RFC itself for full legal notices.";

       revision 2019-01-15 {
         description
           "Initial version";
         reference
           "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
       }

       // identities

       identity hash-algorithm {
         description
           "A base identity for hash algorithm verification";
       }

       identity sha-256 {
         base "hash-algorithm";
         description "The SHA-256 algorithm.";
         reference "RFC 6234: US Secure Hash Algorithms.";
       }

       // typedefs

       typedef cms {
         type binary;
         description
           "A ContentInfo structure, as specified in RFC 5652,
            encoded using ASN.1 distinguished encoding rules (DER),
            as specified in ITU-T X.690.";
         reference
           "RFC 5652:
              Cryptographic Message Syntax (CMS)
            ITU-T X.690:
              Information technology - ASN.1 encoding rules:
              Specification of Basic Encoding Rules (BER),
              Canonical Encoding Rules (CER) and Distinguished
              Encoding Rules (DER).";
```

```
     }

     // yang-data

     rc:yang-data "conveyed-information" {
       choice information-type {
         mandatory true;
         description
           "This choice statement ensures the response contains
            redirect-information or onboarding-information.";
         container redirect-information {
           description
             "Redirect information is described in Section 2.1 in
              RFC XXXX.  Its purpose is to redirect a device to
              another bootstrap server.";
           reference
             "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
           list bootstrap-server {
             key "address";
             min-elements 1;
             description
               "A bootstrap server entry.";
             leaf address {
               type inet:host;
               mandatory true;
               description
                "The IP address or hostname of the bootstrap server the
                 device should redirect to.";
             }
             leaf port {
               type inet:port-number;
               default "443";
               description
                "The port number the bootstrap server listens on.  If no
                 port is specified, the IANA-assigned port for 'https'
                 (443) is used.";
             }
             leaf trust-anchor {
               type cms;
               description
                 "A CMS structure that MUST contain the chain of
                  X.509 certificates needed to authenticate the TLS
                  certificate presented by this bootstrap server.

                  The CMS MUST only contain a single chain of
                  certificates.  The bootstrap server MUST only
                  authenticate to last intermediate CA certificate
                  listed in the chain.
                  In all cases, the chain MUST include a self-signed
                  root certificate.  In the case where the root
                  certificate is itself the issuer of the bootstrap
                  server's TLS certificate, only one certificate
                  is present.

                  If needed by the device, this CMS structure MAY
                  also contain suitably fresh revocation objects
                  with which the device can verify the revocation
                  status of the certificates.

                  This CMS encodes the degenerate form of the SignedData
                  structure that is commonly used to disseminate X.509
                  certificates and revocation objects (RFC 5280).";
               reference
                 "RFC 5280:
                    Internet X.509 Public Key Infrastructure Certificate
                    and Certificate Revocation List (CRL) Profile.";
             }
           }
         }
         container onboarding-information {
           description
             "Onboarding information is described in Section 2.2 in
              RFC XXXX.  Its purpose is to provide the device everything
              it needs to bootstrap itself.";
           reference
             "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
           container boot-image {
             description
               "Specifies criteria for the boot image the device MUST
                be running, as well as information enabling the device
                to install the required boot image.";
             leaf os-name {
               type string;
               description
                 "The name of the operating system software the device
                  MUST be running in order to not require a software
                  image upgrade (ex. VendorOS).";
             }
             leaf os-version {
               type string;
               description
                 "The version of the operating system software the
                  device MUST be running in order to not require a
                  software image upgrade (ex. 17.3R2.1).";
             }
             leaf-list download-uri {
               type inet:uri;
               ordered-by user;
               description
                 "An ordered list of URIs to where the same boot image
                  file may be obtained.  How the URI schemes (http, ftp,
                  etc.) a device supports are known is vendor specific.
                  If a secure scheme (e.g., https) is provided, a device
                  MAY establish an untrusted connection to the remote
                  server, by blindly accepting the server’s end-entity
                  certificate, to obtain the boot image.";
             }
             list image-verification {
               must ’../download-uri’ {
                 description
                   "Download URIs must be provided if an image is to
                    be verified.";
               }
               key hash-algorithm;
               description
                 "A list of hash values that a device can use to verify
                  boot image files with.";
               leaf hash-algorithm {
                 type identityref {
                   base "hash-algorithm";
                 }
                 description
                   "Identifies the hash algorithm used.";
               }
               leaf hash-value {
                 type yang:hex-string;
                 mandatory true;
                 description
                   "The hex-encoded value of the specified hash
                    algorithm over the contents of the boot image
                    file.";
               }
             }
           }
           leaf configuration-handling {
             type enumeration {
               enum "merge" {
                 description
                   "Merge configuration into the running datastore.";
               }
               enum "replace" {
                 description
                   "Replace the existing running datastore with the
                    passed configuration.";
```

```
                }
              }
              must '../configuration';
              description
                "This enumeration indicates how the server should process
                 the provided configuration.";
            }
            leaf pre-configuration-script {
              type script;
              description
                "A script that, when present, is executed before the
                 configuration has been processed.";
            }
            leaf configuration {
              type binary;
              must '../configuration-handling';
              description
                "Any configuration known to the device.  The use of
                 the 'binary' type enables e.g., XML-content to be
                 embedded into a JSON document.  The exact encoding
                 of the content, as with the scripts, is vendor
                 specific.";
            }
            leaf post-configuration-script {
              type script;
              description
                "A script that, when present, is executed after the
                 configuration has been processed.";
            }
          }
        }
      }

      typedef script {
        type binary;
        description
          "A device specific script that enables the execution of
           commands to perform actions not possible thru configuration
           alone.

           No attempt is made to standardize the contents, running
           context, or programming language of the script, other than
           that it can indicate if any warnings or errors occurred and
           can emit output.  The contents of the script are considered
           specific to the vendor, product line, and/or model of the
           device.

           If the script execution indicates that an warning occurred,
           then the device MUST assume that the script had a soft error
           that the script believes will not affect manageability.

           If the script execution indicates that an error occurred,
           the device MUST assume the script had a hard error that the
           script believes will affect manageability.  In this case,
           the script is required to gracefully exit, removing any
           state that might hinder the device’s ability to continue
           the bootstrapping sequence (e.g., process onboarding
           information obtained from another bootstrap server).";
      }
    }
    <CODE ENDS>
```

7. The SZTP Bootstrap Server API

This section defines the API for bootstrap servers. The API is defined as that produced by a RESTCONF [RFC8040] server that supports the YANG 1.1 [RFC7950] module defined in this section.

7.1. API Overview

The following tree diagram provides an overview for the bootstrap server RESTCONF API.

```
   module: ietf-sztp-bootstrap-server

     rpcs:
       +---x get-bootstrapping-data
       |  +---w input
       |  |  +---w signed-data-preferred?   empty
       |  |  +---w hw-model?                string
       |  |  +---w os-name?                 string
       |  |  +---w os-version?              string
       |  |  +---w nonce?                   binary
       |  +--ro output
       |     +--ro reporting-level?    enumeration {onboarding-server}?
       |     +--ro conveyed-information    cms
       |     +--ro owner-certificate?      cms
       |     +--ro ownership-voucher?      cms
       +---x report-progress {onboarding-server}?
          +---w input
             +---w progress-type          enumeration
             +---w message?               string
             +---w ssh-host-keys
             |  +---w ssh-host-key* []
             |     +---w algorithm    string
             |     +---w key-data     binary
             +---w trust-anchor-certs
                +---w trust-anchor-cert*   cms
```

7.2. Example Usage

This section presents three examples illustrating the bootstrap server's API. Two examples are provided for the "get-bootstrapping-data" RPC (once to an untrusted bootstrap server, and again to a trusted bootstrap server), and one example for the "report-progress" RPC.

The following example illustrates a device using the API to fetch its bootstrapping data from a untrusted bootstrap server. In this example, the device sends the "signed-data-preferred" input parameter and receives signed data in the response.

```
REQUEST

[Note: '\' line wrapping for formatting only]

POST /restconf/operations/ietf-sztp-bootstrap-server:get-bootstrappi\
ng-data HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml

<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <signed-data-preferred/>
</input>

RESPONSE

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<output
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <conveyed-information>base64encodedvalue==</conveyed-information>
  <owner-certificate>base64encodedvalue==</owner-certificate>
  <ownership-voucher>base64encodedvalue==</ownership-voucher>
</output>
```

The following example illustrates a device using the API to fetch its bootstrapping data from a trusted bootstrap server. In this example, the device sends addition input parameters to the bootstrap server, which it may use when formulating its response to the device.

```
REQUEST

[Note: '\' line wrapping for formatting only]

POST /restconf/operations/ietf-sztp-bootstrap-server:get-bootstrappi\
ng-data HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml

<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <hw-model>model-x</hw-model>
  <os-name>vendor-os</os-name>
  <os-version>17.3R2.1</os-version>
  <nonce>extralongbase64encodedvalue=</nonce>
</input>

RESPONSE

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<output
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <reporting-level>verbose</reporting-level>
  <conveyed-information>base64encodedvalue==</conveyed-information>
</output>
```

The following example illustrates a device using the API to post a progress report to a bootstrap server. Illustrated below is the "bootstrap-complete" message, but the device may send other progress reports to the server while bootstrapping. In this example, the device is sending both its SSH host keys and a TLS server certificate, which the bootstrap server may, for example, pass to an NMS, as discussed in Appendix C.3.

REQUEST

[Note: ’\’ line wrapping for formatting only]

```
POST /restconf/operations/ietf-sztp-bootstrap-server:report-progress\
 HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml

<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server">
  <progress-type>bootstrap-complete</progress-type>
  <message>example message</message>
  <ssh-host-keys>
    <ssh-host-key>
      <algorithm>ssh-rsa</algorithm>
      <key-data>base64encodedvalue==</key-data>
    </ssh-host-key>
    <ssh-host-key>
      <algorithm>rsa-sha2-256</algorithm>
      <key-data>base64encodedvalue==</key-data>
    </ssh-host-key>
  </ssh-host-keys>
  <trust-anchor-certs>
    <trust-anchor-cert>base64encodedvalue==</trust-anchor-cert>
  </trust-anchor-certs>
</input>
```

RESPONSE

```
HTTP/1.1 204 No Content
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
```

7.3. YANG Module

The bootstrap server’s device-facing API is normatively defined by the YANG module defined in this section.

This module uses data types defined in [RFC4253], [RFC5652], [RFC5280], [RFC6960], and [RFC8366], uses an encoding defined in [ITU.X690.2015], and makes a reference to [RFC4250] and [RFC6187].

```
<CODE BEGINS> file "ietf-sztp-bootstrap-server@2019-01-15.yang"
module ietf-sztp-bootstrap-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server";
  prefix sztp-svr;
```

```
     organization
       "IETF NETCONF (Network Configuration) Working Group";

     contact
       "WG Web:   <http://tools.ietf.org/wg/netconf/>
        WG List:  <mailto:netconf@ietf.org>
        Author:   Kent Watsen <mailto:kwatsen@juniper.net>";

     description
      "This module defines an interface for bootstrap servers, as
       defined by RFC XXXX: Secure Zero Touch Provisioning (SZTP).

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
       'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
       'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
       are to be interpreted as described in BCP 14 (RFC 2119,
       RFC 8174) when, and only when, they appear in all
       capitals, as shown here.

       Copyright (c) 2019 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 (http://trustee.ietf.org/license-info)

       This version of this YANG module is part of RFC XXXX; see the
       RFC itself for full legal notices.";

     revision 2019-01-15 {
       description
         "Initial version";
       reference
         "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
     }

     // features
     feature redirect-server {
       description
        "The server supports being a 'redirect server'.";
     }

     feature onboarding-server {
       description
        "The server supports being an 'onboarding server'.";
     }
     // typedefs

     typedef cms {
       type binary;
       description
         "A CMS structure, as specified in RFC 5652, encoded using
          ASN.1 distinguished encoding rules (DER), as specified in
          ITU-T X.690.";
       reference
         "RFC 5652:
            Cryptographic Message Syntax (CMS)
          ITU-T X.690:
            Information technology - ASN.1 encoding rules:
            Specification of Basic Encoding Rules (BER),
            Canonical Encoding Rules (CER) and Distinguished
            Encoding Rules (DER).";
     }

     // RPCs

     rpc get-bootstrapping-data {
       description
         "This RPC enables a device, as identified by the RESTCONF
          username, to obtain bootstrapping data that has been made
          available for it.";
       input {
         leaf signed-data-preferred {
           type empty;
           description
             "This optional input parameter enables a device to
              communicate to the bootstrap server that it prefers
              to receive signed data.  Devices SHOULD always send
              this parameter when the bootstrap server is untrusted.
              Upon receiving this input parameter, the bootstrap
              server MUST return either signed data, or unsigned
              redirect information; the bootstrap server MUST NOT
              return unsigned onboarding information.";
         }
         leaf hw-model {
           type string;
           description
             "This optional input parameter enables a device to
              communicate to the bootstrap server its vendor specific
              hardware model number.  This parameter may be needed,
              for instance, when a device's IDevID certificate does
              not include the 'hardwareModelName' value in its
              subjectAltName field, as is allowed by 802.1AR-2009.";
           reference
```

```
             "IEEE 802.1AR-2009: IEEE Standard for Local and
                metropolitan area networks - Secure Device Identity";
         }
         leaf os-name {
           type string;
           description
             "This optional input parameter enables a device to
              communicate to the bootstrap server the name of its
              operating system.  This parameter may be useful if
              the device, as identified by its serial number, can
              run more than one type of operating system (e.g.,
              on a white-box system.";
         }
         leaf os-version {
           type string;
           description
             "This optional input parameter enables a device to
              communicate to the bootstrap server the version of its
              operating system.  This parameter may be used by a
              bootstrap server to return an operating system specific
              response to the device, thus negating the need for a
              potentially expensive boot-image update.";
         }
         leaf nonce {
           type binary {
             length "16..32";
           }
           description
             "This optional input parameter enables a device to
              communicate to the bootstrap server a nonce value.
              This may be especially useful for devices lacking
              an accurate clock, as then the bootstrap server
              can dynamically obtain from the manufacturer a
              voucher with the nonce value in it, as described
              in RFC 8366.";
           reference
             "RFC 8366:
                A Voucher Artifact for Bootstrapping Protocols";
         }
       }
       output {
         leaf reporting-level {
           if-feature onboarding-server;
           type enumeration {
             enum standard {
               description
                 "Send just the progress reports required by RFC XXXX.";
               reference
```

```
                 "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
             }
             enum verbose {
               description
                 "Send additional progress reports that might help
                  troubleshooting an SZTP bootstrapping issue.";
             }
           }
           default standard;
           description
             "Specifies the reporting level for progress reports the
              bootstrap server would like to receive when processing
              onboarding information.  Progress reports are not sent
              when processing redirect information, or when the
              bootstrap server is untrusted (e.g., device sent the
              '<signed-data-preferred>' input parameter).";
         }
         leaf conveyed-information {
           type cms;
           mandatory true;
           description
             "An SZTP conveyed information artifact, as described in
              Section 3.1 of RFC XXXX.";
           reference
             "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
         }
         leaf owner-certificate {
           type cms;
           must '../ownership-voucher' {
             description
               "An ownership voucher must be present whenever an owner
                certificate is presented.";
           }
           description
             "An owner certificate artifact, as described in Section
              3.2 of RFC XXXX.  This leaf is optional because it is
              only needed when the conveyed information artifact is
              signed.";
           reference
             "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
         }
         leaf ownership-voucher {
           type cms;
           must '../owner-certificate' {
             description
               "An owner certificate must be present whenever an
                ownership voucher is presented.";
           }
```

```
          description
            "An ownership voucher artifact, as described by Section
             3.3 of RFC XXXX.  This leaf is optional because it is
             only needed when the conveyed information artifact is
             signed.";
          reference
            "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
        }
      }
    }

    rpc report-progress {
      if-feature onboarding-server;
      description
        "This RPC enables a device, as identified by the RESTCONF
         username, to report its bootstrapping progress to the
         bootstrap server.  This RPC is expected to be used when
         the device obtains onboarding-information from a trusted
         bootstap server.";
      input {
        leaf progress-type {
          type enumeration {
            enum "bootstrap-initiated" {
              description
                "Indicates that the device just used the
                 'get-bootstrapping-data' RPC.  The 'message' node
                 below MAY contain any additional information that
                 the manufacturer thinks might be useful.";
            }
            enum "parsing-initiated" {
              description
                "Indicates that the device is about to start parsing
                 the onboarding information.  This progress type is
                 only for when parsing is implemented as a distinct
                 step.";
            }
            enum "parsing-warning" {
              description
                "Indicates that the device had a non-fatal error when
                 parsing the response from the bootstrap server.  The
                 'message' node below SHOULD indicate the specific
                 warning that occurred.";
            }
            enum "parsing-error" {
              description
                "Indicates that the device encountered a fatal error
                 when parsing the response from the bootstrap server.
                 For instance, this could be due to malformed encoding,
                 the device expecting signed data when only unsigned
                 data is provided, the ownership voucher not listing
                 the device's serial number, or because the signature
                 didn't match.  The 'message' node below SHOULD
                 indicate the specific error.  This progress type
                 also indicates that the device has abandoned trying
                 to bootstrap off this bootstrap server.";
            }
            enum "parsing-complete" {
              description
                "Indicates that the device successfully completed
                 parsing the onboarding information.  This progress
                 type is only for when parsing is implemented as a
                 distinct step.";
            }
            enum "boot-image-initiated" {
              description
                "Indicates that the device is about to start
                 processing the boot-image information.";
            }
            enum "boot-image-warning" {
              description
                "Indicates that the device encountered a non-fatal
                 error condition when trying to install a boot-image.
                 A possible reason might include a need to reformat a
                 partition causing loss of data.  The 'message' node
                 below SHOULD indicate any warning messages that were
                 generated.";
            }
            enum "boot-image-error" {
              description
                "Indicates that the device encountered an error when
                 trying to install a boot-image, which could be for
                 reasons such as a file server being unreachable,
                 file not found, signature mismatch, etc.  The
                 'message' node SHOULD indicate the specific error
                 that occurred.  This progress type also indicates
                 that the device has abandoned trying to bootstrap
                 off this bootstrap server.";
            }
            enum "boot-image-mismatch" {
              description
                "Indicates that the device that has determined that
                 it is not running the correct boot image.  This
                 message SHOULD precipitate trying to download
                 a boot image.";
            }
            enum "boot-image-installed-rebooting" {
              description
                "Indicates that the device successfully installed
                 a new boot image and is about to reboot.  After
                 sending this progress type, the device is not
                 expected to access the bootstrap server again
                 for this bootstrapping attempt.";
            }
            enum "boot-image-complete" {
              description
                "Indicates that the device believes that it is
                 running the correct boot-image.";
            }
            enum "pre-script-initiated" {
              description
                "Indicates that the device is about to execute the
                 'pre-configuration-script'.";
            }
            enum "pre-script-warning" {
              description
                "Indicates that the device obtained a warning from the
                 'pre-configuration-script' when it was executed.  The
                 'message' node below SHOULD capture any output the
                 script produces.";
            }
            enum "pre-script-error" {
              description
                "Indicates that the device obtained an error from the
                 'pre-configuration-script' when it was executed. The
                 'message' node below SHOULD capture any output the
                 script produces.  This progress type also indicates
                 that the device has abandoned trying to bootstrap
                 off this bootstrap server.";
            }
            enum "pre-script-complete" {
              description
                "Indicates that the device successfully executed the
                 'pre-configuration-script'.";
            }
            enum "config-initiated" {
              description
                "Indicates that the device is about to commit the
                 initial configuration.";
            }
            enum "config-warning" {
              description
                "Indicates that the device obtained warning messages
                 when it committed the initial configuration.  The
                 'message' node below SHOULD indicate any warning
```

```
                  messages that were generated.";
             }
             enum "config-error" {
               description
                 "Indicates that the device obtained error messages
                  when it committed the initial configuration.  The
                  'message' node below SHOULD indicate the error
                  messages that were generated.  This progress type
                  also indicates that the device has abandoned trying
                  to bootstrap off this bootstrap server.";
             }
             enum "config-complete" {
               description
                 "Indicates that the device successfully committed
                  the initial configuration.";
             }
             enum "post-script-initiated" {
               description
                 "Indicates that the device is about to execute the
                  'post-configuration-script'.";
             }
             enum "post-script-warning" {
               description
                 "Indicates that the device obtained a warning from the
                  'post-configuration-script' when it was executed. The
                  'message' node below SHOULD capture any output the
                  script produces.";
             }
             enum "post-script-error" {
               description
                 "Indicates that the device obtained an error from the
                  'post-configuration-script' when it was executed. The
                  'message' node below SHOULD capture any output the
                  script produces.  This progress type also indicates
                  that the device has abandoned trying to bootstrap
                  off this bootstrap server.";
             }
             enum "post-script-complete" {
               description
                 "Indicates that the device successfully executed the
                  'post-configuration-script'.";
             }
             enum "bootstrap-warning" {
              description
                "Indicates that a warning condition occurred for which
                 there no other 'progress-type' enumeration is deemed
                 suitable.  The 'message' node below SHOULD describe
                 the warning.";
             }
             enum "bootstrap-error" {
              description
                "Indicates that an error condition occurred for which
                 there no other 'progress-type' enumeration is deemed
                 suitable.  The 'message' node below SHOULD describe
                 the error.  This progress type also indicates that
                 the device has abandoned trying to bootstrap off
                 this bootstrap server.";
             }
             enum "bootstrap-complete" {
               description
                 "Indicates that the device successfully processed
                  all 'onboarding-information' provided, and that it
                  is ready to be managed.  The 'message' node below
                  MAY contain any additional information that the
                  manufacturer thinks might be useful.  After sending
                  this progress type, the device is not expected to
                  access the bootstrap server again.";
             }
             enum "informational" {
               description
                 "Indicates any additional information not captured
                  by any of the other progress types. For instance,
                  a message indicating that the device is about to
                  reboot after having installed a boot-image could
                  be provided.  The 'message' node below SHOULD
                  contain information that the manufacturer thinks
                  might be useful.";
             }
           }
           mandatory true;
           description
             "The type of progress report provided.";
         }
         leaf message {
           type string;
           description
             "An optional arbitrary value.";
         }
         container ssh-host-keys {
           when "../progress-type = 'bootstrap-complete'" {
             description
               "SSH host keys are only sent when the progress type
                is 'bootstrap-complete'.";
           }
           description
             "A list of SSH host keys an NMS may use to authenticate
```

```
              subsequent SSH-based connections to this device (e.g.,
              netconf-ssh, netconf-ch-ssh).";
           list ssh-host-key {
             description
               "An SSH host key an NMS may use to authenticate
                subsequent SSH-based connections to this device
                (e.g., netconf-ssh, netconf-ch-ssh).";
             reference
               "RFC 4253: The Secure Shell (SSH) Transport Layer
                          Protocol";
             leaf algorithm {
               type string;
               mandatory true;
               description
                 "The public key algorithm name for this SSH key.

                  Valid values are listed in the 'Public Key Algorithm
                  Names' subregistry of the 'Secure Shell (SSH) Protocol
                  Parameters' registry maintained by IANA.";
               reference
                 "RFC 4250: The Secure Shell (SSH) Protocol Assigned
                            Numbers
                  IANA URL: https://www.iana.org/assignments/ssh-param\\
                            eters/ssh-parameters.xhtml#ssh-parameters-19
                            ('\\' added for formatting reasons)";
             }
             leaf key-data {
               type binary;
               mandatory true;
               description
                 "The binary public key data for this SSH key, as
                  specified by RFC 4253, Section 6.6, i.e.:

                    string    certificate or public key format
                              identifier
                    byte[n]   key/certificate data.";
               reference
                 "RFC 4253: The Secure Shell (SSH) Transport Layer
                            Protocol";
             }
           }
         }
         container trust-anchor-certs {
           when "../progress-type = 'bootstrap-complete'" {
             description
               "Trust anchors are only sent when the progress type
                is 'bootstrap-complete'.";
           }
           description
             "A list of trust anchor certificates an NMS may use to
              authenticate subsequent certificate-based connections
              to this device (e.g., restconf-tls, netconf-tls, or
              even netconf-ssh with X.509 support from RFC 6187).
              In practice, trust anchors for IDevID certificates do
              not need to be conveyed using this mechanism.";
           reference
             "RFC 6187:
                X.509v3 Certificates for Secure Shell Authentication.";
           leaf-list trust-anchor-cert {
             type cms;
             description
              "A CMS structure whose top-most content type MUST be the
               signed-data content type, as described by Section 5 in
               RFC 5652.

               The CMS MUST contain the chain of X.509 certificates
               needed to authenticate the certificate presented by
               the device.

               The CMS MUST contain only a single chain of
               certificates.  The last certificate in the chain
               MUST be the issuer for the device's end-entity
               certificate.

               In all cases, the chain MUST include a self-signed
               root certificate.  In the case where the root
               certificate is itself the issuer of the device's
               end-entity certificate, only one certificate is
               present.

               This CMS encodes the degenerate form of the SignedData
               structure that is commonly used to disseminate X.509
               certificates and revocation objects (RFC 5280).";
             reference
               "RFC 5280:
                  Internet X.509 Public Key Infrastructure
                  Certificate and Certificate Revocation List (CRL)
                  Profile.
                RFC 5652:
                  Cryptographic Message Syntax (CMS)";
           }
         }
       }
     }
   }
   <CODE ENDS>
```

8. DHCP Options

This section defines two DHCP options, one for DHCPv4 and one for DHCPv6. These two options are semantically the same, though syntactically different.

8.1. DHCPv4 SZTP Redirect Option

The DHCPv4 SZTP Redirect Option is used to provision the client with one or more URIs for bootstrap servers that can be contacted to attempt further configuration.

```
         DHCPv4 SZTP Redirect Option

          0                             1
          0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
         |   option-code (143)   |     option-length     |
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
         .                                               .
         .    bootstrap-server-list (variable length)    .
         .                                               .
         +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

         * option-code: OPTION_V4_SZTP_REDIRECT (143)
         * option-length: The option length in octets.
         * bootstrap-server-list: A list of servers for the
            client to attempt contacting, in order to obtain
            further bootstrapping data, in the format shown
            in Section 8.3.
```

DHCPv4 Client Behavior

Clients MAY request the OPTION_V4_SZTP_REDIRECT by including its option code in the Parameter Request List (55) in DHCP request messages.

On receipt of a DHCPv4 Reply message which contains the OPTION_V4_SZTP_REDIRECT, the client processes the response according to Section 5.5, with the understanding that the "address" and "port" values are encoded in the URIs.

Any invalid URI entries received in the uri-data field are ignored by the client. If OPTION_V4_SZTP_REDIRECT does not contain at least one valid URI entry in the uri-data field, then the client MUST discard the option.

As the list of URIs may exceed the maximum allowed length of a single DHCPv4 option (255 octets), the client MUST implement [RFC3396], allowing the URI list to be split across a number of OPTION_V4_SZTP_REDIRECT option instances.

DHCPv4 Server Behavior

The DHCPv4 server MAY include a single instance of Option OPTION_V4_SZTP_REDIRECT in DHCP messages it sends. Servers MUST NOT send more than one instance of the OPTION_V4_SZTP_REDIRECT option.

The server's DHCP message MUST contain only a single instance of the OPTION_V4_SZTP_REDIRECT's 'bootstrap-server-list' field. However, the list of URIs in this field may exceed the maximum allowed length of a single DHCPv4 option (per [RFC3396]).

If the length of 'bootstrap-server-list' is small enough to fit into a single instance of OPTION_V4_SZTP_REDIRECT, the server MUST NOT send more than one instance of this option.

If the length of the 'bootstrap-server-list' field is too large to fit into a single option, then OPTION_V4_SZTP_REDIRECT MUST be split into multiple instances of the option according to the process described in [RFC3396].

8.2. DHCPv6 SZTP Redirect Option

The DHCPv6 SZTP Redirect Option is used to provision the client with one or more URIs for bootstrap servers that can be contacted to attempt further configuration.

DHCPv6 SZTP Redirect Option

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |       option-code (136)       |         option-length         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 .           bootstrap-server-list (variable length)             .
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

* option-code: OPTION_V6_SZTP_REDIRECT (136)
* option-length: The option length in octets.
* bootstrap-server-list: A list of servers for the client to attempt contacting, in order to obtain further bootstrapping data, in the format shown in Section 8.3.

DHCPv6 Client Behavior

Clients MAY request the OPTION_V6_SZTP_REDIRECT option, as defined in [RFC8415], Sections 18.2.1, 18.2.2, 18.2.4, 18.2.5, 18.2.6, and 21.7. As a convenience to the reader, we mention here that the client includes requested option codes in the Option Request Option.

On receipt of a DHCPv6 Reply message which contains the OPTION_V6_SZTP_REDIRECT, the client processes the response according to Section 5.5, with the understanding that the "address" and "port" values are encoded in the URIs.

Any invalid URI entries received in the uri-data field are ignored by the client. If OPTION_V6_SZTP_REDIRECT does not contain at least one valid URI entry in the uri-data field, then the client MUST discard the option.

DHCPv6 Server Behavior

Section 18.3 of [RFC8415] governs server operation in regard to option assignment. As a convenience to the reader, we mention here that the server will send a particular option code only if configured with specific values for that option code and if the client requested it.

Option OPTION_V6_SZTP_REDIRECT is a singleton. Servers MUST NOT send more than one instance of the OPTION_V6_SZTP_REDIRECT option.

8.3. Common Field Encoding

Both of the DHCPv4 and DHCPv6 options defined in this section encode a list of bootstrap server URIs. The "URI" structure is a DHCP option that can contain multiple URIs (see [RFC7227], Section 5.7). Each URI entry in the bootstrap-server-list is structured as follows:

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+-+-+-+-+
 |       uri-length              |          URI                     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-...-+-+-+-+-+-+-+
```

* uri-length: 2 octets long, specifies the length of the URI data.
* URI: URI of SZTP bootstrap server.

The URI of the SZTP bootstrap server MUST use the "https" URI scheme defined in Section 2.7.2 of [RFC7230], and MUST be in form "https://<ip-address-or-hostname>[:<port>]".

9. Security Considerations

9.1. Clock Sensitivity

The solution in this document relies on TLS certificates, owner certificates, and ownership vouchers, all of which require an accurate clock in order to be processed correctly (e.g., to test validity dates and revocation status). Implementations SHOULD ensure devices have an accurate clock when shipped from manufacturing facilities, and take steps to prevent clock tampering.

If it is not possible to ensure clock accuracy, it is RECOMMENDED that implementations disable the aspects of the solution having clock sensitivity. In particular, such implementations should assume that TLS certificates, ownership vouchers, and owner certificates never expire and are not revokable. From an ownership voucher perspective, manufacturers SHOULD issue a single ownership voucher for the lifetime of such devices.

Implementations SHOULD NOT rely on NTP for time, as NTP is not a secure protocol at this time. Note, there is an IETF work-in-progress to secure NTP [I-D.ietf-ntp-using-nts-for-ntp].

9.2. Use of IDevID Certificates

IDevID certificates, as defined in [Std-802.1AR-2018], are RECOMMENDED, both for the TLS-level client certificate used by devices when connecting to a bootstrap server, as well as for the device identity certificate used by owners when encrypting the SZTP bootstrapping data artifacts.

9.3. Immutable Storage for Trust Anchors

Devices MUST ensure that all their trust anchor certificates, including those for connecting to bootstrap servers and verifying ownership vouchers, are protected from external modification.

It may be necessary to update these certificates over time (e.g., the manufacturer wants to delegate trust to a new CA). It is therefore expected that devices MAY update these trust anchors when needed through a verifiable process, such as a software upgrade using signed software images.

9.4. Secure Storage for Long-lived Private Keys

Manufacturer-generated device identifiers may have very long lifetimes. For instance, [Std-802.1AR-2018] recommends using the "notAfter" value 99991231235959Z in IDevID certificates. Given the long-lived nature of these private keys, it is paramount that they are stored so as to resist discovery, such as in a secure cryptographic processor, such as a trusted platform module (TPM) chip.

9.5. Blindly Authenticating a Bootstrap Server

This document allows a device to blindly authenticate a bootstrap server's TLS certificate. It does so to allow for cases where the redirect information may be obtained in an unsecured manner, which is desirable to support in some cases.

To compensate for this, this document requires that devices, when connected to an untrusted bootstrap server, assert that data downloaded from the server is signed.

9.6. Disclosing Information to Untrusted Servers

This document allows devices to establish connections to untrusted bootstrap servers. However, since the bootstrap server is untrusted, it may be under the control of an adversary, and therefore devices SHOULD be cautious about the data they send to the bootstrap server in such cases.

Devices send different data to bootstrap servers at each of the protocol layers TCP, TLS, HTTP, and RESTCONF.

At the TCP protocol layer, devices may relay their IP address, subject to network translations. Disclosure of this information is not considered a security risk.

At the TLS protocol layer, devices may use a client certificate to identify and authenticate themselves to untrusted bootstrap servers. At a minimum, the client certificate must disclose the device's serial number, and may disclose additional information such as the device's manufacturer, hardware model, public key, etc. Knowledge of this information may provide an adversary with details needed to launch an attack. It is RECOMMENDED that secrecy of the network constituency is not relied on for security.

At the HTTP protocol layer, devices may use an HTTP authentication scheme to identify and authenticate themselves to untrusted bootstrap servers. At a minimum, the authentication scheme must disclose the device's serial number and, concerningly, may, depending on the authentication mechanism used, reveal a secret that is only supposed to be known to the device (e.g., a password). Devices SHOULD NOT use an HTTP authentication scheme (e.g., HTTP Basic) with an untrusted bootstrap server that reveals a secret that is only supposed to be known to the device.

At the RESTCONF protocol layer, devices use the "get-bootstrapping-data" RPC, but not the "report-progress" RPC, when connected to an untrusted bootstrap server. The "get-bootstrapping-data" RPC allows additional input parameters to be passed to the bootstrap server (e.g., "os-name", "os-version", "hw-model"). It is RECOMMENDED that devices only pass the "signed-data-preferred" input parameter to an untrusted bootstrap server. While it is okay for a bootstrap server to immediately return signed onboarding information, it is RECOMMENDED that bootstrap servers instead promote the untrusted connection to a trusted connection, as described in Appendix B, thus enabling the device to use the "report-progress" RPC while processing the onboarding information.

9.7. Sequencing Sources of Bootstrapping Data

For devices supporting more than one source for bootstrapping data, no particular sequencing order has to be observed for security reasons, as the solution for each source is considered equally secure. However, from a privacy perspective, it is RECOMMENDED that devices access local sources before accessing remote sources.

9.8. Safety of Private Keys used for Trust

The solution presented in this document enables bootstrapping data to be trusted in two ways, either through transport level security or through the signing of artifacts.

When transport level security (i.e., a trusted bootstrap server) is used, the private key for the end-entity certificate must be online in order to establish the TLS connection.

When artifacts are signed, the signing key is required to be online only when the bootstrap server is returning a dynamically generated signed-data response. For instance, a bootstrap server, upon receiving the "signed-data-preferred" input parameter to the "get-bootstrapping-data" RPC, may dynamically generate a response that is signed.

Bootstrap server administrators are RECOMMENDED to follow best practice to protect the private key used for any online operation. For instance, use of a hardware security module (HSM) is RECOMMENDED. If an HSM is not used, frequent private key refreshes are RECOMMENDED, assuming all bootstrapping devices have an accurate clock (see Section 9.1).

For best security, it is RECOMMENDED that owners only provide bootstrapping data that has been signed, using a protected private key, and encrypted, using the device's public key from its secure device identity certificate.

9.9. Increased Reliance on Manufacturers

The SZTP bootstrapping protocol presented in this document shifts some control of initial configuration away from the rightful owner of the device and towards the manufacturer and its delegates.

The manufacturer maintains the list of well-known bootstrap servers its devices will trust. By design, if no bootstrapping data is found via other methods first, the device will try to reach out to the well-known bootstrap servers. There is no mechanism to prevent this from occurring other than by using an external firewall to block such connections. Concerns related to trusted bootstrap servers are discussed in Section 9.10.

Similarly, the manufacturer maintains the list of voucher signing authorities its devices will trust. The voucher signing authorities issue the vouchers that enable a device to trust an owner's domain certificate. It is vital that manufacturers ensure the integrity of these voucher signing authorities, so as to avoid incorrect assignments.

Operators should be aware that this system assumes that they trust all the pre-configured bootstrap servers and voucher signing authorities designated by the manufacturers. While operators may use points in the network to block access to the well-known bootstrap servers, operators cannot prevent voucher signing authorities from generating vouchers for their devices.

9.10. Concerns with Trusted Bootstrap Servers

Trusted bootstrap servers, whether well-known or discovered, have the potential to cause problems, such as the following.

- A trusted bootstrap server that has been compromised may be modified to return unsigned data of any sort. For instance, a bootstrap server that is only suppose to return redirect information might be modified to return onboarding information. Similarly, a bootstrap server that is only supposed to return signed data, may be modified to return unsigned data. In both cases, the device will accept the response, unaware that it wasn't supposed to be any different. It is RECOMMENDED that maintainers of trusted bootstrap servers ensure that their systems are not easily compromised and, in case of compromise, have mechanisms in place to detect and remediate the compromise as expediently as possible.

- A trusted bootstrap server hosting either unsigned, or signed but not encrypted, data may disclose information to unwanted parties (e.g., an administrator of the bootstrap server). This is a privacy issue only, but could reveal information that might be used in a subsequent attack. Disclosure of redirect information has limited exposure (it is just a list of bootstrap servers), whereas disclosure of onboarding information could be highly revealing (e.g., network topology, firewall policies, etc.). It is RECOMMENDED that operators encrypt the bootstrapping data when its contents are considered sensitive, even to the point of hiding it from the administrators of the bootstrap server, which may be maintained by a 3rd-party.

9.11. Validity Period for Conveyed Information

The conveyed information artifact does not specify a validity period. For instance, neither redirect information nor onboarding information enable "not-before" or "not-after" values to be specified, and neither artifact alone can be revoked.

For unsigned data provided by an untrusted source of bootstrapping data, it is not meaningful to discuss its validity period when the information itself has no authenticity and may have come from anywhere.

For unsigned data provided by a trusted source of bootstrapping data (i.e., a bootstrap server), the availability of the data is the only measure of it being current. Since the untrusted data comes from a trusted source, its current availability is meaningful and, since bootstrap servers use TLS, the contents of the exchange cannot be modified or replayed.

For signed data, whether provided by an untrusted or trusted source of bootstrapping data, the validity is constrained by the validity of the both the ownership voucher and owner certificate used to authenticate it.

The ownership voucher’s validity is primarily constrained by the ownership voucher’s "created-on" and "expires-on" nodes. While [RFC8366] recommends short-lived vouchers (see Section 6.1), the "expires-on" node may be set to any point in the future, or omitted altogether to indicate that the voucher never expires. The ownership voucher’s validity is secondarily constrained by the manufacturer’s PKI used to sign the voucher; whilst an ownership voucher cannot be revoked directly, the PKI used to sign it may be.

The owner certificate’s validity is primarily constrained by the X.509’s validity field, the "notBefore" and "notAfter" values, as specified by the certificate authority that signed it. The owner certificate’s validity is secondarily constrained by the validity of the PKI used to sign the voucher. Owner certificates may be revoked directly.

For owners that wish to have maximum flexibility in their ability to specify and constrain the validity of signed data, it is RECOMMENDED that a unique owner certificate is created for each signed artifact. Not only does this enable a validity period to be specified, for each artifact, but it also enables to the validity of each artifact to be revoked.

9.12. Cascading Trust via Redirects

Redirect Information (Section 2.1), by design, instructs a bootstrapping device to initiate a HTTPS connection to the specified bootstrap servers.

When the redirect information is trusted, the redirect information can encode a trust anchor certificate used by the device to authenticate the TLS end-entity certificate presented by each bootstrap server.

As a result, any compromise in an interaction providing redirect information may result in compromise of all subsequent interactions.

9.13. Possible Reuse of Private Keys

This document describes two uses for secure device identity certificates.

The primary use is for when the device authenticates itself to a bootstrap server, using its private key for TLS-level client-certificate based authentication.

A secondary use is for when the device needs to decrypt provided bootstrapping artifacts, using its private key to decrypt the data or, more precisely, per Section 6 in [RFC5652], decrypt a symmetric key used to decrypt the data.

This document, in Section 3.4 allows for the possibility that the same secure device identity certificate is used for both uses, as [Std-802.1AR-2018] states that a DevID certificate MAY have the "keyEncipherment" KeyUsage bit, in addition to the "digitalSignature" KeyUsage bit, set.

While it is understood that it is generally frowned upon to reuse private keys, this document views such reuse acceptable as there are not any known ways to cause a signature made in one context to be (mis)interpreted as valid in the other context.

9.14. Non-Issue with Encrypting Signed Artifacts

This document specifies the encryption of signed objects, as opposed to the signing of encrypted objects, as might be expected given well-publicized oracle attacks (e.g., the padding oracle attack).

This document does not view such attacks as feasible in the context of the solution because the decrypted text never leaves the device.

9.15. The "ietf-sztp-conveyed-info" YANG Module

The ietf-sztp-conveyed-info module defined in this document defines a data structure that is always wrapped by a CMS structure. When accessed by a secure mechanism (e.g., protected by TLS), then the CMS structure may be unsigned. However, when accessed by an insecure mechanism (e.g., removable storage device), then the CMS structure must be signed, in order for the device to trust it.

Implementations should be aware that signed bootstrapping data only protects the data from modification, and that the contents are still visible to others. This doesn’t affect security so much as privacy. That the contents may be read by unintended parties when accessed by insecure mechanisms is considered next.

The ietf-sztp-conveyed-info module defines a top-level "choice" statement that declares the contents are either "redirect-information" or "onboarding-information". Each of these two cases are now considered.

When the content of the CMS structure is redirect-information, an observer can learn about the bootstrap servers the device is being directed to, their IP addresses or hostnames, ports, and trust anchor certificates. Knowledge of this information could provide an observer some insight into a network’s inner structure.

When the content of the CMS structure is onboarding information, an observer could learn considerable information about how the device is to be provisioned. This information includes the operating system version, initial configuration, and script contents. This information should be considered sensitive and precautions should be taken to protect it (e.g., encrypt the artifact using the device’s public key).

9.16. The "ietf-sztp-bootstrap-server" YANG Module

The ietf-sztp-bootstrap-server module defined in this document specifies an API for a RESTCONF [RFC8040]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

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

This module presents no data nodes (only RPCs). There is no need to discuss the sensitivity of data nodes.

This module defines two RPC operations that may be considered sensitive in some network environments. These are the operations and their sensitivity/vulnerability:

get-bootstrapping-data: This RPC is used by devices to obtain their bootstrapping data. By design, each device, as identified by its authentication credentials (e.g. client certificate), can only obtain its own data. NACM is not needed to further constrain access to this RPC.

report-progress: This RPC is used by devices to report their bootstrapping progress. By design, each device, as identified by its authentication credentials (e.g. client certificate), can only report data for itself. NACM is not needed to further constrain access to this RPC.

10. IANA Considerations

10.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688] maintained at https://www.iana.org/assignments/xml-registry/xml-registry.xhtml#ns. Following the format in [RFC3688], the following registrations are requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-sztp-conveyed-info
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

10.2. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020] maintained at https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml. Following the format defined in [RFC6020], the below registrations are requested:

```
  name:      ietf-sztp-conveyed-info
  namespace: urn:ietf:params:xml:ns:yang:ietf-sztp-conveyed-info
  prefix:    sztp-info
  reference: RFC XXXX

  name:      ietf-sztp-bootstrap-server
  namespace: urn:ietf:params:xml:ns:yang:ietf-sztp-bootstrap-server
  prefix:    sztp-svr
  reference: RFC XXXX
```

10.3. The SMI Security for S/MIME CMS Content Type Registry

This document registers two SMI security codes in the "SMI Security for S/MIME CMS Content Type" registry (1.2.840.113549.1.9.16.1) maintained at https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#security-smime-1. Following the format used in Section 3.4 of [RFC7107], the below registrations are requested:

Decimal	Description	References
TBD1	id-ct-sztpConveyedInfoXML	[RFCXXXX]
TBD2	id-ct-sztpConveyedInfoJSON	[RFCXXXX]

id-ct-sztpConveyedInfoXML indicates that the "conveyed-information" is encoded using XML. id-ct-sztpConveyedInfoJSON indicates that the "conveyed-information" is encoded using JSON.

10.4. The BOOTP Manufacturer Extensions and DHCP Options Registry

This document registers one DHCP code point in the "BOOTP Manufacturer Extensions and DHCP Options" registry maintained at http://www.iana.org/assignments/bootp-dhcp-parameters. Following the format used by other registrations, the below registration is requested:

```
          Tag:         143
          Name:        OPTION_V4_SZTP_REDIRECT
          Data Length: N
          Meaning:     This option provides a list of URIs
                       for SZTP bootstrap servers
          Reference:   [RFCXXXX]
```

Note: this request is to make permanent a previously registered early code point allocation.

10.5. The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Registry

This document registers one DHCP code point in "Option Codes" subregistry of the "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)" registry maintained at http://www.iana.org/assignments/dhcpv6-parameters. Following the format used by other registrations, the below registration is requested:

```
Value:            136
Description:      OPTION_V6_SZTP_REDIRECT
Client ORO:       Yes
Singleton Option: Yes
Reference:        [RFCXXXX]
```

Note: this request is to make permanent a previously registered early code point allocation.

10.6. The Service Name and Transport Protocol Port Number Registry

This document registers one service name in the Service Name and Transport Protocol Port Number Registry [RFC6335] maintained at https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. Following the format defined in Section 8.1.1 of [RFC6335], the below registration is requested:

```
Service Name:           sztp
Transport Protocol(s):   TCP
Assignee:               IESG <iesg@ietf.org>
Contact:                IETF Chair <chair@ietf.org>
Description:            This service name is used to construct the
                        SRV service label "_sztp" for discovering
                        SZTP bootstrap servers.
Reference:              [RFCXXXX]
Port Number:            N/A
Service Code:           N/A
Known Unauthorized Uses: N/A
Assignment Notes:       This protocol uses HTTPS as a substrate.
```

10.7. The DNS Underscore Global Scoped Entry Registry

This document registers one service name in the DNS Underscore Global Scoped Entry Registry [I-D.ietf-dnsop-attrleaf] maintained at TBD_IANA_URL. Following the format defined in Section 4.3 of [I-D.ietf-dnsop-attrleaf], the below registration is requested:

RR Type:	TXT
_NODE NAME:	_sztp
Reference:	[RFCXXXX]

11. References

11.1. Normative References

[I-D.ietf-dnsop-attrleaf]
Crocker, D., "DNS Scoped Data Through "Underscore" Naming of Attribute Leaves", draft-ietf-dnsop-attrleaf-16 (work in progress), November 2018.

[ITU.X690.2015]
International Telecommunication Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <https://www.itu.int/rec/T-REC-X.690/>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC3396] Lemon, T. and S. Cheshire, "Encoding Long Options in the Dynamic Host Configuration Protocol (DHCPv4)", RFC 3396, DOI 10.17487/RFC3396, November 2002, <https://www.rfc-editor.org/info/rfc3396>.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[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>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

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

[RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, DOI 10.17487/RFC7227, May 2014, <https://www.rfc-editor.org/info/rfc7227>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <https://www.rfc-editor.org/info/rfc8366>.

[RFC8415] Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/info/rfc8415>.

[Std-802.1AR-2018] IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", June 2018, <https://standards.ieee.org/standard/802_1AR-2018.html>.

11.2. Informative References

[I-D.ietf-netconf-crypto-types] Watsen, K. and H. Wang, "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-02 (work in progress), October 2018.

[I-D.ietf-netconf-trust-anchors] Watsen, K., "YANG Data Model for Global Trust Anchors", draft-ietf-netconf-trust-anchors-02 (work in progress), October 2018.

[I-D.ietf-ntp-using-nts-for-ntp] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", draft-ietf-ntp-using-nts-for-ntp-15 (work in progress), December 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>.

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, DOI 10.17487/RFC4250, January 2006, <https://www.rfc-editor.org/info/rfc4250>.

[RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, DOI 10.17487/RFC6187, March 2011, <https://www.rfc-editor.org/info/rfc6187>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[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>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <https://www.rfc-editor.org/info/rfc6698>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <https://www.rfc-editor.org/info/rfc6891>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[RFC7107] Housley, R., "Object Identifier Registry for the S/MIME Mail Security Working Group", RFC 7107, DOI 10.17487/RFC7107, January 2014, <https://www.rfc-editor.org/info/rfc7107>.

[RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <https://www.rfc-editor.org/info/rfc7766>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <https://www.rfc-editor.org/info/rfc8071>.

[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>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Appendix A. Example Device Data Model

This section defines a non-normative data model that enables the configuration of SZTP bootstrapping and discovery of what parameters are used by a device's bootstrapping logic.

A.1. Data Model Overview

The following tree diagram provides an overview for the SZTP device data model.

```
 module: example-device-data-model
   +--rw sztp
      +--rw enabled?                          boolean
      +--ro idevid-certificate?               ct:end-entity-cert-cms
      |       {bootstrap-servers}?
      +--ro bootstrap-servers {bootstrap-servers}?
      |  +--ro bootstrap-server* [address]
      |     +--ro address    inet:host
      |     +--ro port?      inet:port-number
      +--ro bootstrap-server-trust-anchors {bootstrap-servers}?
      |  +--ro reference*   ta:pinned-certificates-ref
      +--ro voucher-trust-anchors {signed-data}?
         +--ro reference*   ta:pinned-certificates-ref
```

In the above diagram, notice that there is only one configurable node "enabled". The expectation is that this node would be set to "true" in device's factory default configuration and that it would either be set to "false" or deleted when the SZTP bootstrapping is longer needed.

A.2. Example Usage

Following is an instance example for this data model.

```
<sztp xmlns="https://example.com/sztp-device-data-model">
  <enabled>true</enabled>
  <idevid-certificate>base64encodedvalue==</idevid-certificate>
  <bootstrap-servers>
    <bootstrap-server>
      <address>sztp1.example.com</address>
      <port>8443</port>
    </bootstrap-server>
    <bootstrap-server>
      <address>sztp2.example.com</address>
      <port>8443</port>
    </bootstrap-server>
    <bootstrap-server>
      <address>sztp3.example.com</address>
      <port>8443</port>
    </bootstrap-server>
  </bootstrap-servers>
  <bootstrap-server-trust-anchors>
    <reference>manufacturers-root-ca-certs</reference>
  </bootstrap-server-trust-anchors>
  <voucher-trust-anchors>
    <reference>manufacturers-root-ca-certs</reference>
  </voucher-trust-anchors>
</sztp>
```

A.3. YANG Module

The device model is defined by the YANG module defined in this section.

This module uses data types defined in [RFC6991], [I-D.ietf-netconf-crypto-types], and [I-D.ietf-netconf-trust-anchors].

```
module example-device-data-model {
  yang-version 1.1;
  namespace "https://example.com/sztp-device-data-model";
  prefix sztp-ddm;

  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types";
  }

  import ietf-crypto-types {
    prefix ct;
    revision-date 2018-06-04;
    description
```

```
         "This revision is defined in the -00 version of
          draft-ietf-netconf-crypto-types";
        reference
         "draft-ietf-netconf-crypto-types:
            Common YANG Data Types for Cryptography";
      }

      import ietf-trust-anchors {
        prefix ta;
        revision-date 2018-06-04;
        description
         "This revision is defined in -00 version of
          draft-ietf-netconf-trust-anchors.";
        reference
         "draft-ietf-netconf-trust-anchors:
            YANG Data Model for Global Trust Anchors";
      }

      organization
        "Example Corporation";

      contact
        "Author: Bootstrap Admin <mailto:admin@example.com>";

      description
        "This module defines a data model to enable SZTP
         bootstrapping and discover what parameters are used.
         This module assumes the use of an IDevID certificate,
         as opposed to any other client certificate, or the
         use of an HTTP-based client authentication scheme.";

      revision 2019-01-15 {
        description
          "Initial version";
        reference
          "RFC XXXX: Secure Zero Touch Provisioning (SZTP)";
      }

      // features

      feature bootstrap-servers {
        description
          "The device supports bootstrapping off bootstrap servers.";
      }

      feature signed-data {
        description
          "The device supports bootstrapping off signed data.";
```

```
     }

     // protocol accessible nodes

     container sztp {
       description
         "Top-level container for SZTP data model.";
       leaf enabled {
         type boolean;
         default false;
         description
           "The 'enabled' leaf controls if SZTP bootstrapping is
            enabled or disabled.  The default is 'false' so that, when
            not enabled, which is most of the time, no configuration
            is needed.";
       }
       leaf idevid-certificate {
         if-feature bootstrap-servers;
         type ct:end-entity-cert-cms;
         config false;
         description
           "This CMS structure contains the IEEE 802.1AR-2009
            IDevID certificate itself, and all intermediate
            certificates leading up to, and optionally including,
            the manufacturer's well-known trust anchor certificate
            for IDevID certificates.  The well-known trust anchor
            does not have to be a self-signed certificate.";
         reference
           "IEEE 802.1AR-2009:
              IEEE Standard for Local and metropolitan area
              networks - Secure Device Identity.";
       }
       container bootstrap-servers {
         if-feature bootstrap-servers;
         config false;
         description
           "List of bootstrap servers this device will attempt
            to reach out to when bootstrapping.";
         list bootstrap-server {
           key "address";
           description
             "A bootstrap server entry.";
           leaf address {
             type inet:host;
             mandatory true;
             description
               "The IP address or hostname of the bootstrap server the
                device should redirect to.";
           }
           leaf port {
             type inet:port-number;
             default "443";
             description
               "The port number the bootstrap server listens on.  If no
                port is specified, the IANA-assigned port for 'https'
                (443) is used.";
           }
         }
       }
       container bootstrap-server-trust-anchors {
         if-feature bootstrap-servers;
         config false;
         description "Container for a list of trust anchor references.";
         leaf-list reference {
           type ta:pinned-certificates-ref;
           description
             "A reference to a list of pinned certificate authority (CA)
              certificates that the device uses to validate bootstrap
              servers with.";
         }
       }
       container voucher-trust-anchors {
         if-feature signed-data;
         config false;
         description "Container for a list of trust anchor references.";
         leaf-list reference {
           type ta:pinned-certificates-ref;
           description
             "A reference to a list of pinned certificate authority (CA)
              certificates that the device uses to validate ownership
              vouchers with.";
         }
       }
     }
   }
```

Appendix B. Promoting a Connection from Untrusted to Trusted

The following diagram illustrates a sequence of bootstrapping activities that promote an untrusted connection to a bootstrap server to a trusted connection to the same bootstrap server. This enables a device to limit the amount of information it might disclose to an adversary hosting an untrusted bootstrap server.

```
                                                         +----------+
                                                         |Deployment|
                                                         | Specific |
   +------+                                              |Bootstrap |
   |Device|                                              |  Server  |
   +------+                                              +----------+
      |                                                        |
      | 1. "HTTPS" Request ("signed-data-preferred", nonce)    |
      |------------------------------------------------------->|
      | 2. "HTTPS" Response (signed redirect information)      |
      |<-------------------------------------------------------|
      |                                                        |
      |                                                        |
      | 3. HTTPS Request (os-name=xyz, os-version=123, etc.)   |
      |------------------------------------------------------->|
      | 4. HTTPS Response (unsigned onboarding information     |
      |<-------------------------------------------------------|
      |                                                        |
```

The interactions in the above diagram are described below.

1. The device initiates an untrusted connection to a bootstrap server, as is indicated by putting "HTTPS" in double quotes above. It is still an HTTPS connection, but the device is unable to authenticate the bootstrap server’s TLS certificate. Because the device is unable to trust the bootstrap server, it sends the "signed-data-preferred" input parameter, and optionally also the "nonce" input parameter, in the "get-bootstrapping-data" RPC. The "signed-data-preferred" parameter informs the bootstrap server that the device does not trust it and may be holding back some additional input parameters from the server (e.g., other input parameters, progress reports, etc.). The "nonce" input parameter enables the bootstrap server to dynamically obtain an ownership voucher from a MASA, which may be important for devices that do not have a reliable clock.

2. The bootstrap server, seeing the "signed-data-preferred" input parameter, knows that it can either send unsigned redirect information or signed data of any type. But, in this case, the bootstrap server has the ability to sign data and chooses to respond with signed redirect information, not signed onboarding information as might be expected, securely redirecting the device back to it again. Not displayed but, if the "nonce" input parameter was passed, the bootstrap server could dynamically connect to a download a voucher from the MASA having the nonce value in it. Details regarding a protocol enabling this integration is outside the scope of this document.

3. Upon validating the signed redirect information, the device establishes a secure connection to the bootstrap server. Unbeknownst to the device, it is the same bootstrap server it was connected to previously but, because the device is able to authenticate the bootstrap server this time, it sends its normal "get-bootstrapping-data" request (i.e., with additional input parameters) as well as its progress reports (not depicted).

4. This time, because the "signed-data-preferred" parameter was not passed, having access to all of the device's input parameters, the bootstrap server returns, in this example, unsigned onboarding information to the device. Note also that, because the bootstrap server is now trusted, the device will send progress reports to the server.

Appendix C. Workflow Overview

The solution presented in this document is conceptualized to be composed of the non-normative workflows described in this section. Implementation details are expected to vary. Each diagram is followed by a detailed description of the steps presented in the diagram, with further explanation on how implementations may vary.

C.1. Enrollment and Ordering Devices

The following diagram illustrates key interactions that may occur from when a prospective owner enrolls in a manufacturer's SZTP program to when the manufacturer ships devices for an order placed by the prospective owner.

```
                                  +-----------+
   +------------+                 |Prospective|                    +---+
   |Manufacturer|                 |   Owner   |                    |NMS|
   +------------+                 +-----------+                    +---+
         |                              |                            |
         |                              |                            |
         |  1. initiate enrollment      |                            |
         #<-----------------------------|                            |
         #                              |                            |
         #                              |                            |
         #     IDevID trust anchor      |                            |
         #----------------------------->#  set IDevID trust anchor   |
         #                              #--------------------------->|
         #                              |                            |
         #     bootstrap server         |                            |
         #     account credentials      |                            |
         #----------------------------->#  set credentials           |
         |                              #--------------------------->|
         |                              |                            |
         |                              |                            |
         |  2. set owner certificate trust anchor                    |
         |<----------------------------------------------------------|
         |                              |                            |
         |                              |                            |
         |  3. place device order       |                            |
         |<-----------------------------#  model devices             |
         |                              #--------------------------->|
         |                              |                            |
         |  4. ship devices and send    |                            |
         |     device identifiers and   |                            |
         |     ownership vouchers       |                            |
         |----------------------------->#  set device identifiers    |
         |                              #  and ownership vouchers    |
         |                              #--------------------------->|
         |                              |                            |
```

Each numbered item below corresponds to a numbered item in the diagram above.

1. A prospective owner of a manufacturer's devices initiates an enrollment process with the manufacturer. This process includes the following:

   * Regardless how the prospective owner intends to bootstrap their devices, they will always obtain from the manufacturer the trust anchor certificate for the IDevID certificates. This certificate will is installed on the prospective owner's

NMS so that the NMS can authenticate the IDevID certificates when they are presented to subsequent steps.

* If the manufacturer hosts an Internet based bootstrap server (e.g., a redirect server) such as described in Section 4.4, then credentials necessary to configure the bootstrap server would be provided to the prospective owner. If the bootstrap server is configurable through an API (outside the scope of this document), then the credentials might be installed on the prospective owner's NMS so that the NMS can subsequently configure the manufacturer-hosted bootstrap server directly.

2. If the manufacturer's devices are able to validate signed data (Section 5.4), and assuming that the prospective owner's NMS is able to prepare and sign the bootstrapping data itself, the prospective owner's NMS might set a trust anchor certificate onto the manufacturer's bootstrap server, using the credentials provided in the previous step. This certificate is the trust anchor certificate that the prospective owner would like the manufacturer to place into the ownership vouchers it generates, thereby enabling devices to trust the owner's owner certificate. How this trust anchor certificate is used to enable devices to validate signed bootstrapping data is described in Section 5.4.

3. Some time later, the prospective owner places an order with the manufacturer, perhaps with a special flag checked for SZTP handling. At this time, or perhaps before placing the order, the owner may model the devices in their NMS, creating virtual objects for the devices with no real-world device associations. For instance the model can be used to simulate the device's location in the network and the configuration it should have when fully operational.

4. When the manufacturer fulfills the order, shipping the devices to their intended locations, they may notify the owner of the devices' serial numbers and shipping destinations, which the owner may use to stage the network for when the devices power on. Additionally, the manufacturer may send one or more ownership vouchers, cryptographically assigning ownership of those devices to the owner. The owner may set this information on their NMS, perhaps binding specific modeled devices to the serial numbers and ownership vouchers.

C.2. Owner Stages the Network for Bootstrap

The following diagram illustrates how an owner might stage the network for bootstrapping devices.

```
               +----------+ +------------+
               |Deployment| |Manufacturer| +------+ +------+
               | Specific | |   Hosted   | | Local| | Local| +---------+
         +---+ |Bootstrap | | Bootstrap  | |  DNS | | DHCP | |Removable|
         |NMS| |  Server  | |   Server   | |Server| |Server| | Storage |
         +---+ +----------+ +------------+ +------+ +------+ +---------+
           |        |             |            |        |         |
   1.      |        |             |            |        |         |
   activate|        |             |            |        |         |
   modeled |        |             |            |        |         |
   device  |        |             |            |        |         |
   ------->|        |             |            |        |         |
           | 2. (optional)        |            |        |         |
           |    configure         |            |        |         |
           |    bootstrap         |            |        |         |
           |    server            |            |        |         |
           |------->|             |            |        |         |
           |        |             |            |        |         |
           | 3. (optional) configure           |        |         |
           |    bootstrap server  |            |        |         |
           |--------------------->|            |        |         |
           |        |             |            |        |         |
           |        |             |            |        |         |
           | 4. (optional) configure DNS server|        |         |
           |---------------------------------->|        |         |
           |        |             |            |        |         |
           |        |             |            |        |         |
           | 5. (optional) configure DHCP server        |         |
           |------------------------------------------->|         |
           |        |             |            |        |         |
           |        |             |            |        |         |
           | 6. (optional) store bootstrapping artifacts on media |
           |----------------------------------------------------->|
           |        |             |            |        |         |
           |        |             |            |        |         |
```

Each numbered item below corresponds to a numbered item in the diagram above.

1. Having previously modeled the devices, including setting their fully operational configurations and associating device serial numbers and (optionally) ownership vouchers, the owner might "activate" one or more modeled devices. That is, the owner tells the NMS to perform the steps necessary to prepare for when the real-world devices power up and initiate the bootstrapping process. Note that, in some deployments, this step might be combined with the last step from the previous workflow. Here it

is depicted that an NMS performs the steps, but they may be performed manually or through some other mechanism.

2. If it is desired to use a deployment-specific bootstrap server, it must be configured to provide the bootstrapping data for the specific devices. Configuring the bootstrap server may occur via a programmatic API not defined by this document. Illustrated here as an external component, the bootstrap server may be implemented as an internal component of the NMS itself.

3. If it is desired to use a manufacturer hosted bootstrap server, it must be configured to provide the bootstrapping data for the specific devices. The configuration must be either redirect or onboarding information. That is, either the manufacturer hosted bootstrap server will redirect the device to another bootstrap server, or provide the device with the onboarding information itself. The types of bootstrapping data the manufacturer hosted bootstrap server supports may vary by implementation; some implementations may only support redirect information, or only support onboarding information, or support both redirect and onboarding information. Configuring the bootstrap server may occur via a programmatic API not defined by this document.

4. If it is desired to use a DNS server to supply bootstrapping data, a DNS server needs to be configured. If multicast DNS-SD is desired, then the DNS server must reside on the local network, otherwise the DNS server may reside on a remote network. Please see Section 4.2 for more information about how to configure DNS servers. Configuring the DNS server may occur via a programmatic API not defined by this document.

5. If it is desired to use a DHCP server to supply bootstrapping data, a DHCP server needs to be configured. The DHCP server may be accessed directly or via a DHCP relay. Please see Section 4.3 for more information about how to configure DHCP servers. Configuring the DHCP server may occur via a programmatic API not defined by this document.

6. If it is desired to use a removable storage device (e.g., USB flash drive) to supply bootstrapping data, the data would need to be placed onto it. Please see Section 4.1 for more information about how to configure a removable storage device.

C.3. Device Powers On

The following diagram illustrates the sequence of activities that occur when a device powers on.

```
                                                     +----------+
                                      +-----------+  |Deployment|
                                      | Source of |  | Specific |
   +------+                           | Bootstrap |  |Bootstrap |  +---+
   |Device|                           |   Data    |  |  Server  |  |NMS|
   +------+                           +-----------+  +----------+  +---+
      |                                     |              |         |
      |                                     |              |         |
      | 1. if SZTP bootstrap service        |              |         |
      |    is not enabled, then exit.       |              |         |
      |                                     |              |         |
      | 2. for each source supported, check |              |         |
      |    for bootstrapping data.          |              |         |
      |------------------------------------>|              |         |
      |                                     |              |         |
      | 3. if onboarding information found, |              |         |
      |    initialize self and, only if     |              |         |
      |    source is a trusted bootstrap    |              |         |
      |    server, send progress reports.   |              |         |
      |------------------------------------>#              |         |
      |                                     # webhook      |         |
      |                                     #----------------------->|
      |                                                    |         |
      | 4. else if redirect-information found, for each    |         |
      |    bootstrap server specified, check for data.     |         |
      |-+------------------------------------------------->|         |
      | |                                                  |         |
      | | if more redirect-information is found, recurse   |         |
      | | (not depicted), else if onboarding information   |         |
      | | found, initialize self and post progress reports |         |
      | +------------------------------------------------->#         |
      |                                                    # webhook |
      |                                                    #-------->|
      |
      | 5. retry sources and/or wait for manual provisioning.
      |
```

The interactions in the above diagram are described below.

1. Upon power being applied, the device checks to see if SZTP bootstrapping is configured, such as must be the case when running its "factory default" configuration. If SZTP bootstrapping is not configured, then the bootstrapping logic exits and none of the following interactions occur.

2. For each source of bootstrapping data the device supports, preferably in order of closeness to the device (e.g., removable

storage before Internet based servers), the device checks to see if there is any bootstrapping data for it there.

3. If onboarding information is found, the device initializes itself accordingly (e.g., installing a boot-image and committing an initial configuration). If the source is a bootstrap server, and the bootstrap server can be trusted (i.e., TLS-level authentication), the device also sends progress reports to the bootstrap server.

   * The contents of the initial configuration should configure an administrator account on the device (e.g., username, SSH public key, etc.), and should configure the device either to listen for NETCONF or RESTCONF connections or to initiate call home connections [RFC8071], and should disable the SZTP bootstrapping service (e.g., the "enabled" leaf in data model presented in Appendix A).

   * If the bootstrap server supports forwarding device progress reports to external systems (e.g., via a webhook), a "bootstrap-complete" progress report (Section 7.3) informs the external system to know when it can, for instance, initiate a connection to the device. To support this scenario further, the "bootstrap-complete" progress report may also relay the device's SSH host keys and/or TLS certificates, with which the external system can use to authenticate subsequent connections to the device.

   If the device successfully completes the bootstrapping process, it exits the bootstrapping logic without considering any additional sources of bootstrapping data.

4. Otherwise, if redirect information is found, the device iterates through the list of specified bootstrap servers, checking to see if the bootstrap server has bootstrapping data for the device. If the bootstrap server returns more redirect information, then the device processes it recursively. Otherwise, if the bootstrap server returns onboarding information, the device processes it following the description provided in (3) above.

5. After having tried all supported sources of bootstrapping data, the device may retry again all the sources and/or provide manageability interfaces for manual configuration (e.g., CLI, HTTP, NETCONF, etc.). If manual configuration is allowed, and such configuration is provided, the configuration should also disable the SZTP bootstrapping service, as the need for bootstrapping would no longer be present.

Appendix D. Change Log

D.1. ID to 00

- Major structural update; the essence is the same. Most every section was rewritten to some degree.

- Added a Use Cases section

- Added diagrams for "Actors and Roles" and "NMS Precondition" sections, and greatly improved the "Device Boot Sequence" diagram

- Removed support for physical presence or any ability for configlets to not be signed.

- Defined the Conveyed Information DHCP option

- Added an ability for devices to also download images from configuration servers

- Added an ability for configlets to be encrypted

- Now configuration servers only have to support HTTP/S - no other schemes possible

D.2. 00 to 01

- Added boot-image and validate-owner annotations to the "Actors and Roles" diagram.

- Fixed 2nd paragraph in section 7.1 to reflect current use of anyxml.

- Added encrypted and signed-encrypted examples

- Replaced YANG module with XSD schema

- Added IANA request for the Conveyed Information DHCP Option

- Added IANA request for media types for boot-image and configuration

D.3. 01 to 02

- Replaced the need for a configuration signer with the ability for each NMS to be able to sign its own configurations, using manufacturer signed ownership vouchers and owner certificates.

- Renamed configuration server to bootstrap server, a more representative name given the information devices download from it.

- Replaced the concept of a configlet by defining a southbound interface for the bootstrap server using YANG.

- Removed the IANA request for the boot-image and configuration media types

D.4. 02 to 03

- Minor update, mostly just to add an Editor's Note to show how this draft might integrate with the draft-pritikin-anima-bootstrapping-keyinfra.

D.5. 03 to 04

- Major update formally introducing unsigned data and support for Internet-based redirect servers.

- Added many terms to Terminology section.

- Added all new "Guiding Principles" section.

- Added all new "Sources for Bootstrapping Data" section.

- Rewrote the "Interactions" section and renamed it "Workflow Overview".

D.6. 04 to 05

- Semi-major update, refactoring the document into more logical parts

- Created new section for information types

- Added support for DNS servers

- Now allows provisional TLS connections

- Bootstrapping data now supports scripts

- Device Details section overhauled

- Security Considerations expanded

- Filled in enumerations for notification types

D.7. 05 to 06

- Minor update

- Added many Normative and Informative references.

- Added new section Other Considerations.

D.8. 06 to 07

- Minor update

- Added an Editorial Note section for RFC Editor.

- Updated the IANA Considerations section.

D.9. 07 to 08

- Minor update

- Updated to reflect review from Michael Richardson.

D.10. 08 to 09

- Added in missing "Signature" artifact example.

- Added recommendation for manufacturers to use interoperable formats and file naming conventions for removable storage devices.

- Added configuration-handling leaf to guide if config should be merged, replaced, or processed like an edit-config/yang-patch document.

- Added a pre-configuration script, in addition to the post-configuration script from -05 (issue #15).

D.11. 09 to 10

- Factored ownership voucher and voucher revocation to a separate document: draft-kwatsen-netconf-voucher. (issue #11)

- Removed <configuration-handling> options "edit-config" and "yang-patch". (issue #12)

- Defined how a signature over signed-data returned from a bootstrap server is processed. (issue #13)

- Added recommendation for removable storage devices to use open/standard file systems when possible. (issue #14)

- Replaced notifications "script-[warning/error]" with "[pre/post]-script-[warning/error]". (goes with issue #15)

- switched owner-certificate to be encoded using the PKCS #7 format. (issue #16)

- Replaced md5/sha1 with sha256 inside a choice statement, for future extensibility. (issue #17)

- A ton of editorial changes, as I went thru the entire draft with a fine-toothed comb.

D.12. 10 to 11

- fixed yang validation issues found by IETFYANGPageCompilation. note: these issues were NOT found by pyang --ietf or by the submission-time validator...

- fixed a typo in the yang module, someone the config false statement was removed.

D.13. 11 to 12

- fixed typo that prevented Appendix B from loading the examples correctly.

- fixed more yang validation issues found by IETFYANGPageCompilation. note: again, these issues were NOT found by pyang --ietf or by the submission-time validator...

- updated a few of the notification enumerations to be more consistent with the other enumerations (following the warning/error pattern).

- updated the information-type artifact to state how it is encoded, matching the language that was in Appendix B.

D.14. 12 to 13

- defined a standalone artifact to encode the old information-type into a PKCS #7 structure.

- standalone information artifact hardcodes JSON encoding (to match the voucher draft).

- combined the information and signature PKCS #7 structures into a single PKCS #7 structure.

- moved the certificate-revocations into the owner-certificate's PKCS #7 structure.

- eliminated support for voucher-revocations, to reflect the voucher-draft's switch from revocations to renewals.

D.15. 13 to 14

- Renamed "bootstrap information" to "onboarding information".

- Rewrote DHCP sections to address the packet-size limitation issue, as discussed in Chicago.

- Added Ian as an author for his text-contributions to the DHCP sections.

- Removed the Guiding Principles section.

D.16. 14 to 15

- Renamed action "notification" to "update-progress" and, likewise "notification-type" to "update-type".

- Updated examples to use "base64encodedvalue==" for binary values.

- Greatly simplified the "Artifact Groupings" section, and moved it as a subsection to the "Artifacts" section.

- Moved the "Workflow Overview" section to the Appendix.

- Renamed "bootstrap information" to "update information".

- Removed "Other Considerations" section.

- Tons of editorial updates.

D.17. 15 to 16

- tweaked language to refer to "initial state" rather than "factory default configuration", so as accommodate white-box scenarios.

- added a paragraph to Intro regarding how the solution primarily regards physical machines, but could be extended to VMs by a future document.

- added a pointer to the Workflow Overview section (recently moved to the Appendix) to the Intro.

- added a note that, in order to simplify the verification process, the "Conveyed Information" PKCS #7 structure MUST also contain the signing X.509 certificate.

- noted that the owner certificate's must either have no Key Usage or the Key Usage must set the "digitalSignature" bit.

- noted that the owner certificate's subject and subjectAltName values are not constrained.

- moved/consolidated some text from the Artifacts section down to the Device Details section.

- tightened up some ambiguous language, for instance, by referring to specific leaf names in the Voucher artifact.

- reverted a previously overzealous s/unique-id/serial-number/ change.

- modified language for when ZTP runs from when factory-default config is running to when ZTP is configured, which the factory-defaults should set .

D.18. 16 to 17

- Added an example for how to promote an untrusted connection to a trusted connection.

- Added a "query parameters" section defining some parameters enabling scenarios raised in last call.

- Added a "Disclosing Information to Untrusted Servers" section to the Security Considerations.

D.19. 17 to 18

- Added Security Considerations for each YANG module.

- Reverted back to the device always sending its DevID cert.

- Moved data tree to "get-bootstrapping-data" RPC.

- Moved the "update-progress" action to a "report-progress" RPC.

- Added an "signed-data-preferred" parameter to "get-bootstrapping-data" RPC.

- Added the "ietf-zerotouch-device" module.

- Lots of small updates.

D.20. 18 to 19

- Fixed "must" expressions, by converting "choice" to a "list" of "image-verification", each of which now points to a base identity called "hash-algorithm". There's just one algorithm currently defined (sha-256). Wish there was a standard crypto module that could identify such identities.

D.21. 19 to 20

- Now references I-D.ietf-netmod-yang-tree-diagrams.

- Fixed tree-diagrams in Section 2 to always reflect current YANG (now they are now dynamically generated).

- The "redirect-information" container's "trust-anchor" is now a CMS structure that can contain a chain of certificates, rather than a single certificate.

- The "onboarding-information" container's support for image verification reworked to be extensible.

- Added a reference to the "Device Details" section to the new example-device-data-model module.

- Clarified that the device must always pass its IDevID certificate, even for untrusted bootstrap servers.

- Fixed the description statement for the "script" typedef to refer to the [pre/post]-script-[warning/error] enums, rather than the legacy script-[warning/error] enums.

- For the get-bootstrapping-data RPC's input, removed the "remote-id" and "circuit-id" fields, and added a "hw-model" field.

- Improved DHCP error handling text.

- Added MUST requirement for DHCPv6 client and server implementing [RFC3396] to handle URI lists longer than 255 octets.

- Changed the "configuration" value in onboarding-information to be type "binary" instead of "anydata".

- Moved everything from PKCS#7 to CMS (this shows up as a big change).

- Added the early code point allocation assignments for the DHCP Options in the IANA Considerations section, and updated the RFC Editor note accordingly.

- Added RFC Editor request to replace the assigned values for the CMS content types.

- Relaxed auth requirements from device needing to always send IDevID cert to device needing to always send authentication credentials, as this better matches what RFC 8040 Section 2.5 says.

- Moved normative module "ietf-zerotouch-device" to non-normative module "example-device-data-model".

- Updated Title, Abstract, and Introduction per discussion on list.

D.22. 20 to 21

- Now any of the three artifact can be encrypted.

- Fixed some line-too-long issues.

D.23. 21 to 22

- Removed specifics around how scripts indicate warnings or errors and how scripts emit output.

- Moved the SZTP Device Data Model section to the Appendix.

- Modified the YANG module in the SZTP Device Data Model section to reflect the latest trust-anchors and keystore drafts.

- Modified types in other YANG modules to more closely emulate what is in draft-ietf-netconf-crypto-types.

D.24. 22 to 23

- Rewrote section 5.6 (processing onboboarding information) to be clearer about error handling and retained state. Specifically:

  - Clarified that a script, upon having an error, must gracefully exit, cleaning up any state that might hinder subsequent executions.

  - Added ability for scripts to be executed again with a flag enabling them to clean up state from a previous execution.

  - Clarified that the conifguration commit is atomic.

  - Clarified that any error encountered after committing the configuration (e.g., in the "post-configuration-script") must rollback the configuration to the previous configuration.

  - Clarified that failure to successfully deliver the "bootstrap-initiated" and "bootstrap-complete" progress types must be treated as an error.

  - Clarified that "return to bootstrapping sequence" is to be interpreted in the recursive context. Meaning that the device rolls-back one loop, rather than start over from scratch.

- Changed how a device verifies a boot-image from just "MUST match one of the supplied fingerprints" to also allow for the verification to use an cryptographic signature embedded into the image itself.

- Added more "progress-type" enums for visibility reasons, enabling more strongly-typed debug information to be sent to the bootstrap server.

- Added Security Considerations based on early SecDir review.

- Added recommendation for device to send warning if the initial config does not disable the bootstrapping process.

D.25. 23 to 24

- Follow-ups from SecDir and Shepherd.

- Added "boot-image-complete" enumeration.

D.26. 24 to 25

- Removed remaining old "bootstrapping information" term usage.

- Fixed DHCP Option length definition.

- Added reference to RFC 6187.

D.27. 25 to 26

- Updated URI structure text (sec 8.3) and added norm. ref to RFC7230 reflecting Alexey Melnikov's comment.

- Added IANA registration for the 'zerotouch' service, per IESG review from Adam Roach.

- Clarified device's looping behavior and support for alternative provisioning mechanisms, per IESG review from Mirja Kuehlewind.

- Updated "ietf-sztp-bootstrap-server:ssh-host-key" from leaf-list to list, per IESG review from Benjamin Kaduk.

- Added option size text to DHCPv4 option size to address Suresh Krishnan's IESG review discuss point.

- Updated RFC3315 to RFC8415 and associated section references.

- Revamped the DNS Server section, after digging into Alexey Melnikov comment.

- Fixed IETF terminology template section in both YANG modules.

D.28. 26 to 27

- Added Security Consideration for cascading trust via redirects.

- Modified the get-bootstrapping-data RPC's "nonce" input parameter to being a minimum of 16-bytes (used to be 8-bytes).

- Added Security Consideration regarding possible reuse of device's private key.

- Added Security Consideration regarding use of sign-then-encrypt.

- Renamed "Zero Touch"/"zerotouch" throughout. Now uses "SZTP" when referring to the draft/solution, and "conveyed" when referring to the bootstrapping artifact.

- Added missing text for "encrypted unsigned conveyed information" case.

- Renamed "untrusted-connection" input paramter to "signed-data-preferred"

- Switch yd:yang-data back to rc:yang-data

- Added a couple features to the bootstrap-server module.

D.29. 27 to 28

- Modified DNS section to no longer reference DNS-SD (now just plain TXT and SRV lookups, via multicast or unicast.

- Registers "_sztp" in the DNS Underscore Global Scoped Entry Registry.

- Updated 802.1AR reference to current spec version.

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Michael Behringer, Dean Bogdanovic, Martin Bjorklund, Joe Clarke, Dave Crocker, Toerless Eckert, Stephen Farrell, Stephen Hanna, Wes Hardaker, David Harrington, Mirja Kuehlewind, Radek Krejci, Suresh Krishnan, Benjamin Kaduk, David Mandelberg, Alexey Melnikov, Russ Mundy, Reinaldo Penno, Randy Presuhn, Max Pritikin, Michael Richardson, Adam Roach, Phil Shafer, Juergen Schoenwaelder.

Special thanks goes to Steve Hanna, Russ Mundy, and Wes Hardaker for brainstorming the original solution during the IETF 87 meeting in Berlin.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Mikael Abrahamsson
T-Systems

EMail: mikael.abrahamsson@t-systems.se

Ian Farrer
Deutsche Telekom AG

EMail: ian.farrer@telekom.de

NETCONF E. Voit
Internet-Draft A. Clemm
Intended status: Informational A. Tripathy
Expires: April 15, 2016 E. Nilsen-Nygaard
A. Gonzalez Prieto
Cisco Systems
October 13, 2015

Restconf subscription and HTTP push for YANG datastores
draft-voit-restconf-yang-push-00

Abstract

This document defines Restconf subscription and push mechanisms to continuously stream information from YANG datastores over HTTP. These mechanisms allow client applications or operations support systems to request custom sets of updates from a YANG datastore. This document also specifies how to stream updates over HTTP without Restconf. In either case, updates are pushed by a datastore to a receiver per a subscription policy, without requiring continuous requests.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Requirements for subscriptions to YANG datastores are defined in [pub-sub-reqs]. Mechanisms to support YANG subscriptions and datastore object push over a NETCONF are defined in [netconf-yang-push]. Restconf support of subscriptions, with HTTP transport of pushed updates is also needed by the market. This document provides such a specification.

Key benefits of pushing data via HTTP include:

- Ability to configure static subscriptions on a Publisher

- Ability for the Publisher to initiate communications with the Receiver

- Ability of a Subscriber to be different from the Receiver

There are also additional benefits which can be realized when pushing updates via HTTP/2 [RFC7540]:

- Subscription multiplexing over independent HTTP/2 streams

- Stream prioritization

- Stream dependencies

- Flow control on independent streams

- Header compression

These additional benefits will address issues resulting from head-of-line blocking and relative subscription priority.

To maximize transport independence of YANG subscription methods, this document reuses many capabilities of [netconf-yang-push][] including:

- Operations for creating, modifying and deleting subscriptions

- Syntax and parameters for negotiating the subscription

- YANG data model to manage subscriptions

- Mechanisms to communicate subscription filters and data streams

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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

Dynamic YANG Subscription: Subscription negotiated with Publisher via create, modify, and delete control plane signaling messages.

Publisher: an entity responsible for distributing subscribed YANG object data per the terms of a Subscription. In general, a Publisher is the owner of the YANG datastore that is subjected to the Subscription.

Receiver: the target where a Publisher pushes updates. In many deployments, the Receiver and Subscriber will be the same entity.

Static YANG Subscription: A Subscription installed via a configuration interface.

Subscriber: An entity able to request and negotiate a contract for push updates from a Publisher.

Subscription: A contract between a Subscriber and a Publisher, stipulating which information the Receiver wishes to have pushed from the Publisher without the need for further solicitation.

Subscription Update: Set of data nodes and object values pushed together as a unit and intended to meet the obligations of a single subscription at a snapshot in time.

3. Solution

This document specifies mechanisms that allow subscribed information updates to be pushed from a YANG datastore. Subscriptions may either be initiated via requests by Subscribers, or statically configured on a Publisher. As in [netconf-yang-push], Publisher must respond to a subscription request explicitly positively or negatively. Negative responses will include information about why the Subscription was not accepted, in order to facilitate converging on an acceptable set of Subscription parameters.

Once a Subscription has been established, updates are pushed to the Receiver until the Subscription terminates. Based on parameters within the Subscription, these updates can be streamed immediately as any subscribed objects change, or sent periodically.

3.1. Subscription Model

Subscriptions use the base data model from [netconf-yang-push]. This model is extended with several optional parameters for Subscription Priority and Subscription Dependency. These parameters allow a Subscriber or other configuration interface to assert how it prefers the Publisher allocate resources when handling multiple Subscriptions. These parameters are intended to be used in conjunction with the transport layer. Specifically, when a new Subscription is being established with an underlying transport is HTTP/2, these parameters may be directly mapped into HTTP/2 to prioritize transport and to assist with flow control of individual streams.

3.2. Subscription states at Publisher

Below is the state machine for the Publisher. It is important to note that a Subscription doesn’t exist at the Publisher until it is accepted and made active. The assertion of a <create-subscription> by a Subscriber is insufficient for that asserted subscription to be externally visible via this state machine.

Subscription states at Publisher

```
                    .-------.
                    | start |
                    '-------'
                        |
                     create
                        |
                        |   .----------modify-------------.
                        v   v                              '
                  .-----------.                      .-----------.
       .--------. |           |------>suspend------->|           |
     modify      '|  active   |                      | suspended |
       '--------->|           |<----reactivate<------|           |
                  '-----------'                      '-----------'
                        |                                  |
                     delete                             delete
                        |                                  |
                        v                                  |
                    .-------.                              |
                    |  end  |<-----------------------------'
                    '-------'
```

Of interest in this state machine are the following:

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

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

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

3.3. Mechanisms for Subscription Establishment and Maintenance

On a Publisher, it must be possible to instantiate a Subscription via dynamic Subscriber signaling, as well as via Static configuration.

Dynamic YANG Subscriptions are signaled Subscriptions aimed at the running datastore and are unable to impact the startup configuration. They should always terminate when there is loss of transport session connectivity between the Publisher and Receiver.

Static Subscriptions are applied via an operations interface to the startup and running configurations. Loss or non-availability of transport session connectivity will place the Subscription into the suspended state. Logic beyond the scope of this specification will dictate when any particular Subscription should be reactivated. There are three models for Subscription establishment and maintenance:

1. Dynamic YANG Subscription: Subscriber and Receiver are the same

2. Static YANG Subscription

3. Dynamic YANG Subscription: Subscriber and Receiver are different

The first two are described in this section. The third is described in Appendix A. This third option can be moved into the body of this specification should the IETF community desire. In theory, all three models may be intermixed in a single deployment. Figure 2 shows such a scenario.

```
                          .---------------.
                          |   Publisher   |
                          '---------------'
                            ^   ^   |   ^
                            |   |   |   |
          .-----Restconf----'   |   |   '-----Restconf----.
          |               .-----'   '-HTTP-.              |
          V               |                V              |
   .-------------.  .------------.  .----------.  .------------.
   | Subscriber+ |  | Operations |  | Receiver |  | Subscriber |
   | Receiver    |  |  /Config   |  '----------'  '------------'
   '-------------'  '------------'    ^     ^               ^
          ^         (out of scope)    :     :               :
          :             ^             :     :....Model 3....:
        Model 1         :...Model 2...:      (out of scope)
```

3.3.1. Dynamic YANG Subscription: Subscriber and Receiver are the same

With all Dynamic YANG Subscriptions, as with [netconf-yang-push] it must be possible to configure and manage Subscriptions via signaling. This signaling is transported over [restconf]. Once established, streaming Subscription Updates are then delivered via Restconf SSE.

3.3.2. Static YANG Subscription

With a Static YANG Subscription, all information needed to establish a secure object push relationship with that Receiver must be configured via a configuration interface on the Publisher. This information includes all the <create-subscription> information identified in section 3.3.1. This information also includes the Receiver address, encoding selection, and any security credentials required to establish TLS between the Publisher and Receiver. Mechanisms for locally configuring these parameters are outside the scope of this document.

With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing the streaming updates to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that updates be pushed via Restconf. In place of Restconf, a TLS secured HTTP Client connection must be established with an HTTP Server located on the Receiver. Subscription Updates will then be sent via HTTP Post messages to the Receiver.

Post messages will be addressed to HTTP augmentation code on the Receiver capable accepting and responding to Subscription Updates. At least the initial Post message must include the URI for the subscribed resource. This URI can be retained for future use by the Receiver.

After successful receipt of an initial Subscription Update for a particular Subscription, this augmentation should reply back with an HTTP status code of 201 (Created). Further successful receipts should result in the return of code of 202 (Accepted). At any point, receipt of any status codes from 300-510 with the exception of 408 (Request Timeout) should result in the movement of the Subscription to the suspended state. A sequential series of multiple 408 exceptions should also drive the Subscription to a suspended state.

Security on an HTTP client/Publisher can be strengthened by only accepting Response code feedback for recently initiated HTTP POSTs.

Figure 3 depicts this message flow.

```
      +-----------+                 +----------+
      | Publisher |                 | Receiver |
      +-----------+                 +----------+
           |<--------------TLS------------->|
           |                               |
           |HTTP POST (Sub ID, URI, data1) |
           |------------------------------>|
           |              HTTP 201 (Created)|
           |<------------------------------|
           |HTTP POST (Sub ID, data2)      |
           |------------------------------>|
           |     HTTP 200 or 202 (Accepted)|
           |<------------------------------|
           |             data3             |
           |<----------------------------->|
```

If HTTP/2 transport is available to a Receiver, the Publisher should also:

- point individual Subscription Updates to a unique HTTP/2 stream for that Subscription,

- take any subscription-priority and provision it into the HTTP/2 stream priority, and

- take any subscription-dependency and provision it into the HTTP/2 stream dependency.

3.4. Negotiation of Subscription Policies

When using signaling to create a Dynamic YANG Subscription, negotiable parameters will include the same negotiable parameters defined within [netconf-yang-push].

Additionally, negotiation may also include Subscription Priority. A Publisher may accept a Subscriber asserted Priority, as well as rejecting a subscription with a hint at what priority might be accepted.

3.5. Support for Periodic and On-change

Implementations must support periodic and/or on-change subscriptions as defined in [netconf-yang-push].

3.6. Filters and Streams

Implementations must support filters and streams as defined in [netconf-yang-push].

3.7. Authorization

Same authorization model for data as [netconf-yang-push] will be used. This includes functions of the Netconf Access Control Model [RFC6536] applied to objects to be pushed via Restconf.

A Subscription (including a Static YANG Subscription) may only be established if the Subscriber or some entity statically configuring via the Publisher’s operational interface has read access to the target data node.

3.8. Subscription Multiplexing

When pushed directly over HTTP/2, it is expected that each Subscription Update will be allocated a separate Stream. The will enable multiplexing, and address issues of Head-of-line blocking with different priority Subscriptions.

When pushed via Restconf over HTTP/2, different Subscriptions will not be mapped to independent HTTP/2 streams. When Restconf specifies this mapping, it should be integrated into this specification.

Even without HTTP/2 multiplexing, it is possible that updates might be delivered in a different sequence than generated. Reasons for this might include (but are not limited to):

- different durations needed to create various Subscription Updates,

- marshalling and bundling of multiple Subscription Updates for transport, and

- parallel HTTP1.1 sessions

Therefore each Subscription Update will include a microsecond level timestamp to ensure that a receiver understands the time when a that update was generated. Use of this timestamp can give an indication of the state of objects at a Publisher when state-entangled information is received across different subscriptions. The use of the latest Subscription Update timestamp for a particular object update can introduce errors. So when state-entangled updates have inconsistent object values and temporally close timestamps, a Receiver might consider performing a ’get’ to validate the current state of objects.

3.9. Push Data Stream and Transport Mapping

Transported updates will contain data for one or more Subscription Updates. Each transported Subscription Update notification contains several parameters:

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

- Data nodes containing a representation of the datastore subtree containing the updates. The set of data nodes must be filtered per access control rules to contain only data that the subscriber is authorized to see.

- An event time which contains the time stamp at publisher when the event is generated.

3.9.1. Pushing Subscription Updates via Restconf

Subscribers can dynamically learn whether a RESTCONF server supports yang-push. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream push-update. E.g.:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/
         streams/stream=yang-push HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
```

If the server supports it, it may respond

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
<stream xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
            <name>yang-push</name>
            <description>Yang push stream</description>
            <access>
               <encoding>xml</encoding>
               <location>https://example.com/streams/yang-push-xml
               </location>
            </access>
            <access>
               <encoding>json</encoding>
               <location>https://example.com/streams/yang-push-json
               </location>
            </access>
         </stream>
```

If the server does not support yang push, it may respond

```
HTTP/1.1 404 Not Found
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
```

Subscribers can determine the URL to receive updates by sending an HTTP GET request for the "location" leaf with the stream list entry. The stream to use for yang push is the push-update stream. The location returned by the publisher can be used for the actual notification subscription. Note that different encodings are supporting using different locations. For example, he subscriber might send the following request:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/
         streams/stream=yang-push/access=xml/location HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
```

The publisher might send the following response:

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
   <location
        xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
        https://example.com/streams/yang-push-xml
   </location>
```

To subscribe and start receiving updates, the subscriber can then send an HTTP GET request for the URL returned by the publisher in the request above. The accept header must be "text/event -stream". The publisher handles the connection as an event stream, using the Server Sent Events[W3C-20121211] transport strategy.

The publisher MUST support as query parameters for a GET method on this resource all the parameters of a subscription. The only exception is the encoding, which is embedded in the URI. An example of this is:

```
// subtree filter = /foo
// periodic updates, every 5 seconds
GET /mystreams/yang-push?subscription-id=my-sub&period=5&
           xpath-filter=%2Fex:foo[starts-with("bar"."some"]
```

Should the publisher not support the requested subscription, it may reply:

```
HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+xml
    <errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
       <error>
           <error-type>application</error-type>
           <error-tag>operation-not-supported</error-tag>
           <error-severity>error</error-severity>
           <error-message>Xpath filters not supported</error-message>
           <error-info>
               <supported-subscription xmlns="urn:ietf:params:xml:ns:
                   netconf:datastore-push:1.0">
                   <subtree-filter/>
               </supported-subscription>
           </error-info>
       </error>
     </errors>
```

with an equivalent JSON encoding representation of:

```
HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+json
      {
        "ietf-restconf:errors": {
          "error": {
            "error-type": "protocol",
            "error-tag": "operation-not-supported",
            "error-message": "Xpath filters not supported."
            "error-info": {
               "datastore-push:supported-subscription": {
                     "subtree-filter": [null]
                 }
            }
          }
        }
      }
```

The following is an example of a push Subscription Update data for the subscription above. It contains a subtree with root foo that contains a leaf called bar:

XML encoding representation:

```
  <?xml version="1.0" encoding="UTF-8"?>
  <notification xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
     <subscription-id xmlns="urn:ietf:params:xml:ns:restconf:
         datastore-push:1.0">
           my-sub
     </subscription-id>
     <eventTime>2015-03-09T19:14:56Z</eventTime>
     <datastore-contents xmlns="urn:ietf:params:xml:ns:restconf:
        datastore-push:1.0">
        <foo xmlns="http://example.com/yang-push/1.0">
          <bar>some_string</bar>
        </foo>
     </datastore-contents>
  </notification>
```

Or with the equivalent YANG over JSON encoding representation as defined in[yang-json] :

```
{
  "ietf-restconf:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56Z",
    "datastore-push:datastore-contents": {
      "example-mod:foo": { "bar": "some_string" }
    }
  }
}
```

To modify a subscription, the subscriber issues another GET request on the provided URI using the same subscription-id as in the original request. For example, to modify the update period to 10 seconds, the subscriber may send:

```
GET /mystreams/yang-push?subscription-id=my-sub&period=10&
       subtree-filter=%2Ffoo’
```

To delete a subscription, the subscriber issues a DELETE request on the provided URI using the same subscription-id as in the original request

```
DELETE /mystreams/yang-push?subscription-id=my-sub
```

3.9.2. Pushing Subscription Updates directly via HTTP

For any version of HTTP, the basic encoding will look as below is the above JSON representation wrapped in an HTTP header. Mechanism will be

```
   POST (IP+Port) HTTP/1.1
   From: (Identifier for Network Element)
   User-Agent: (CiscoYANGPubSub/1.0)
   Content-Type: multipart/form-data
   Content-Length: (determined runtime)
   {
     "ietf-yangpush:notification": {
       "datastore-push:subscription-id": "my-sub",
       "eventTime": "2015-03-09T19:14:56Z",
       "datastore-push:datastore-contents": {
         "foo": { "bar": "some_string" }
       }
     }
   }
```

3.10. YANG Tree

Below is the object tree for the model. All items are imported from [netconf-yang-push] except for the addition of "subscription-priority" and "subscription-dependency".

```
      module: ietf-restconf-yang-push
      +-ro system-streams
      |  +-ro system-stream*                  system-stream
      +-rw filters
      |  +-rw filter* [filter-id]
      |     +-rw filter-id                    filter-id
      |     +-rw subtree-filter?              subtree-filter
      |     +-rw xpath-filter?                yang:xpath1.0
      +-rw subscription-config
      |  +-rw datastore-push-subscription*   [subscription-id]
      |  +--rw datastore-push-subscription*  [subscription-id]
      |     +--rw subscription-id             subscription-id
      |     +--rw target-datastore?           datastore
      |     +--rw stream?                     system-stream
      |     +--rw encoding?                   encoding
      |     +--rw start-time?                 yang:date-and-time
      |     +--rw stop-time?                  yang:date-and-time
      |     +--rw (update-trigger)?
      |     |  +--:(periodic)
      |     |  |  +--rw period?               yang:timeticks
      |     |  +--:(on-change)
      |     |     +--rw no-synch-on-start?    empty
      |     |     +--rw dampening-period      yang:timeticks
      |     |     +--rw excluded-change*      change-type
      |     +--rw (filterspec)?
      |     |  +--:(inline)
      |     |  |  +--rw subtree-filter?       subtree-filter
```

```
     |     |  |  +--rw xpath-filter?        yang:xpath1.0
     |     |  +--:(by-reference)
     |     |     +--rw filter-ref?          filter-ref
     |     +--rw receiver-address
     |     |  +-rw (push-base-transport)?
     |     |     +-:(tcpudp)
     |     |        +-rw tcpudp
     |     |           +-rw address?        inet:host
     |     |           +-rw port?           inet:port-number
     |     +--rw subscription-priority?     uint8
     |     +--rw subscription-dependency?   string
     +-ro subscriptions
        +--ro datastore-push-subscription* [subscription-id]
           +--ro subscription-id            subscription-id
           +--ro configured-subscription?   empty
           +--ro subscription-status?       identityref
           +--ro target-datastore?          datastore
           +--ro stream?                    system-stream
           +--ro encoding?                  encoding
           +--ro start-time?                yang:date-and-time
           +--ro stop-time?                 yang:date-and-time
           +--ro (update-trigger)?
           |  +--:(periodic)
           |  |  +--ro period?              yang:timeticks
           |  +--:(on-change)
           |     +--ro no-synch-on-start?   empty
           |     +--ro dampening-period     yang:timeticks
           |     +--ro excluded-change*     change-type
           +--ro (filterspec)?
           |  +--:(inline)
           |  |  +--ro subtree-filter?      subtree-filter
           |  |  +--ro xpath-filter?        yang:xpath1.0
           |  +--:(by-reference)
           |     +--ro filter-ref?          filter-ref
           +--ro receiver-address
           |  +--ro (push-base-transport)?
           |     +--:(tcpudp)
           |        +--ro tcpudp
           |           +--ro address?       inet:host
           |           +--ro port?          inet:port-number
           +--rw subscription-priority?     uint8
           +--rw subscription-dependency?   string
```

4. YANG Module

```
   namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-push";

     prefix "rc-push";
     import ietf-datastore-push {
       prefix ds-push;
     }

     organization
      "IETF NETCONF (Network Configuration) Working Group";

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

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

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

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

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

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

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

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

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

     revision 2015-10-01 {
       description
         "Initial revision.";
       reference "restconf YANG Datastore push";
     }

     grouping subscription-qos {
       description
        "This grouping describes Quality of Service information
         concerning a subscription.  This information is passed to lower
         layers for transport priortization and treatment";
       leaf subscription-priority {
         type uint8;
         description
          "Relative priority for a subscription.   Allows an underlying
           transport layer perform informed load balance allocations
           between various subscriptions";
       }
       leaf subscription-dependency {
         type string;
         description
          "Provides the Subscription ID of a parent subscription
           without which this subscription should not exist. In
           other words, there is no reason to stream these objects
           if another subscription is missing.";
       }
     }

     augment "/ds-push:subscription-config/" +
             "ds-push:datastore-push-subscription" {
       description
         "Aguments configured subscriptions with QoS parameters.";
       uses subscription-qos;
     }

     augment "/ds-push:subscriptions/" +
             "ds-push:datastore-push-subscription" {
       description
         "Aguments the list of currently active subscriptions
           with QoS parameters.";
       uses subscription-qos;
     }

     augment "/ds-push:create-subscription/" +
             "ds-push:input" {
       description
         "Aguments the create subscription rpc with QoS parameters.";
       uses subscription-qos;
     }

     augment "/ds-push:modify-subscription/" +
             "ds-push:input" {
       description
         "Aguments the modify subscription rpc with QoS parameters.";
       uses subscription-qos;
     }

   }
```

5. Security Considerations

Subscriptions could be used to intentionally or accidentally overload resources of a Publisher. For this reason, it is important that the Publisher has the ability to prioritize the establishment and push of updates where there might be resource exhaust potential. In addition, a server needs to be able to suspend existing subscriptions when needed. When this occurs, the subscription status must be updated accordingly and the clients are notified.

A Subscription could be used to retrieve data in subtrees that a client has not authorized access to. Therefore it is important that data pushed via a Subscription is authorized equivalently with regular data retrieval operations. Data being pushed to a client needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model [RFC6536] applies.

One or more Publishers could be used to overwhelm a Receiver which doesn’t even support subscriptions. Therefore Updates MUST only be transmittable over Encrypted transports. Clients which do not want pushed data need only terminate or refuse any transport sessions from the Publisher.

One or more Publishers could overwhelm a Receiver which is unable to control or handle the volume of Updates received. In deployments where this might be a concern, transports supporting per-subscription Flow Control and Prioritization (such as HTTP/2) should be selected.

Another benefit is that a well-behaved Publisher implementation is that it is difficult to a Publisher to perform a DoS attack on a Receiver. DoS attack protection comes from:

- the requirement for trust of a TLS session before publication,
- the need for an HTTP transport augmentation on the Receiver, and
- that the Publication process is suspended when the Receiver doesn’t respond.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <http://www.rfc-editor.org/info/rfc6520>.

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

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

6.2. Informative References

[call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", July 2015, <https://tools.ietf.org/html/draft-ietf-netconf-call-home-09>.

[netconf-yang-push]
Clemm, Alexander., Gonzalez Prieto, Alberto., and Eric. Voit, "Subscribing to YANG datastore push updates", October 2015, <https://datatracker.ietf.org/doc/draft-clemm-netconf-yang-push/>.

[pub-sub-reqs]
Voit, Eric., Clemm, Alexander., and Alberto. Gonzalez Prieto, "Subscribing to datastore push updates", October 2015, <https://datatracker.ietf.org/doc/draft-ietf-i2rs-pub-sub-requirements/>.

[restconf]
Bierman, Andy., Bjorklund, Martin., and Kent. Watsen, "RESTCONF Protocol", July 2015, <https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf/>.

[W3C-20121211]
"Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", December 2012, <http://www.w3.org/TR/2012/CR-eventsource-20121211>.

[yang-json]
Lhotka, Ladislav., "JSON Encoding of Data Modeled with YANG", October 2015, <https://datatracker.ietf.org/doc/draft-ietf-netmod-yang-json/>.

Appendix A. Dynamic YANG Subscription when the Subscriber and Receiver are different

The methods of Sections 3.3.1 and 3.3.2 can be combined to enable deployment models where the Subscriber and Receiver are different. Such separation can be useful with some combination of:

- An operator wants any Subscriptions immediately deleted should TLS connectivity be lost. (I.e., Subscriptions don’t default into a ’Suspended’ state on the Publisher.)

- An operator wants the Publisher to include highly restrictive capacity management and security mechanisms outside of domain of existing operational or programmatic interfaces.

- Restconf is not desired on the Receiver.

- The Publisher doesn’t want to maintain Restconf subscriptions with many Receivers.

To do this, first the necessary information must be signaled as part of the <create-subscription>. This includes all the information described in section 3.3.2, with the exception of the security credentials. (It is assumed that any security credentials required for establishing any transport connections are pre-provisioned on all devices.)

Using this set of Subscriber provided information, the same process described within section 3.3.2 will be followed. There is one exception. When an HTTP status code is 201 is received by the Publisher, it will inform the Subscriber of Subscription establishment success via its Restconf connection.

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

Putting it all together, the message flow is:

```
     +------------+          +-----------+            +----------+
     | Subscriber |          | Publisher |            | Receiver |
     +------------+          +-----------+            +----------+
           | Restconf PUT:        |                            |
           | <create-subscription>|                            |
           |--------------------->|                            |
           |                      |                            |
           |                      |<-----------TLS------------>|
           |                      |                            |
           |                      |HTTP POST (Sub ID, data1,   |
           |                      |(stream ID, URI?))          |
           |                      |--------------------------->|
           |                      |          HTTP 201 (Created)|
           |                      |<---------------------------|
           |     Success: HTTP 204|                            |
           |<---------------------|                            |
           |                      |HTTP POST (Sub ID, data2)   |
           |                      |--------------------------->|
           |                      |  HTTP 200 or 202 (Accepted)|
           |                      |<---------------------------|
           |                      |            data3           |
           |                      |<-------------------------->|
           |                      |                            |
```

Appendix B. End-to-End Deployment Guidance

Several technologies are expected to be seen within a deployment to achieve security and ease-of-use requirements. These are not necessary for an implementation of this specification, but will be useful to consider when considering the operational context.

B.1. Call Home

Pub/Sub implementations should have the ability to transparently incorporate lower layer technologies such as Call Home so that secure TLS connections are always originated from the Publisher. There is a Restconf Call home function in [call-home]. For security reasons, this should be implemented as desired.

B.2. TLS Heartbeat

Unlike NETCONF, HTTP sessions might not quickly allow a Subscriber to recognize when the communication path has been lost from the Publisher. To recognize this, it is possible for a Receiver (usually the subscriber) to establish a TLS heartbeat [RFC6520]. In the case where a TLS heartbeat is included, it should be sent just from Receiver to Publisher. Loss of the heartbeat should result in the Subscription being terminated with the Subscriber (even when the

Subscriber and Receiver are different). The Subscriber can then attempt to re-establish the subscription if desired. If the Subscription remains active on the Publisher, future receipt of objects associated with that (or any other unknown) subscription ID should result in a <delete-subscription> being returned to the Publisher from the Receiver.

B.3. Putting it together

If Subscriber and receiver are same entity then subscriber can direct send create_subscription message to publisher. Once the subscription moved to accepted state, the receiver can use Server Sent Events [W3C-20121211] transport strategy to subscriber event notifications for the data as defined in[restconf].

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Cisco Systems

Email: alex@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com