NETCONF
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2016

A. Gonzalez Prieto
A. Clemm
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
S. Chisholm
Ciena
H. Trevino
Cisco Systems
June 15, 2016

Subscribing to YANG-Defined Event Notifications
draft-gonzalez-netconf-5277bis-02

Abstract

This document defines capabilities and operations for providing asynchronous message notification delivery for notifications defined using YANG. Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF and Restconf. The capabilities and operations defined in this document along with their mapping onto NETCONF transport (to be specified in a separate document, but still included in the current document version) are intended to obsolete RFC 5277.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

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

Table of Contents

1. Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service for the NETCONF protocol . This is an optional capability built on top of the base NETCONF definition. This document defines capabilities and operations for providing asynchronous message notification delivery for notifications defined using YANG, including capabilities and operations necessary to establish, monitor, and support subscriptions to notification delivery.

Notification delivery can occur over a variety of protocols used commonly in conjunction with YANG, such as NETCONF [RFC6241] and Restconf [I-D.ietf-netconf-restconf]. The capabilities and operations defined in this document along with their mapping onto NETCONF transport (to be specified in a separate document, but still included in the current document version) are intended to obsolete RFC 5277.

Editor's note: The current version of this document specifies both capabilities and operations for providing asynchronous notification delivery, as well as mapping of those capabilities and operations onto NETCONF. The transport mapping to NETCONF will be moved into a separate document and will be removed in future revisions of this document.

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:

- Each subscription requires a separate NETCONF connection, which is wasteful.
- The only mechanism to terminate a subscription is terminating the underlying NETCONF connection.
- No ability to modify subscriptions once they have been created.
- No ability to notify the receiver of a subscription if the server is dropping events.
- 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.

- o 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.

Configured 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. Configured 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 configured 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 configured subscription is driven by configuration being present on the running configuration. This implies configured subscriptions persist across reboots, and persists even when transport is unavailable. This also means configured subscriptions do not support negotiation.

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

- o 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 configured 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 configured 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 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 server maintains a list of available event streams as operational data. A client can retrieve this list like any other YANG-defined data, for example using the <get> operation when using NETCONF.

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. Filters

A NETCONF Server implementation SHOULD support the ability to perform filtering of notification records per RFC 5277.

2.5. Subscription State Model at the Publisher

Below is the state machine of a subscription 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.

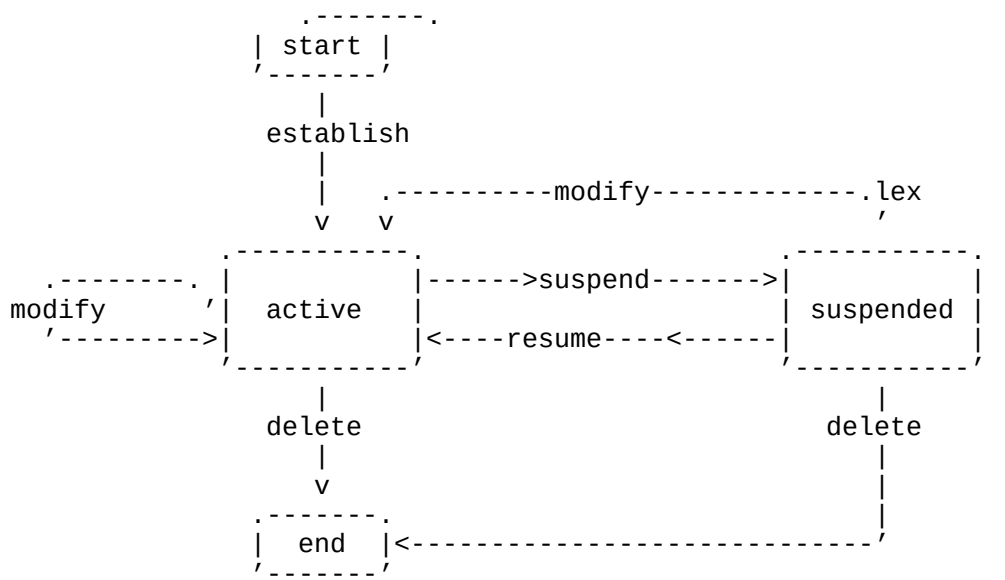


Figure 1: Subscription states at publisher

Of interest in this state machine are the following:

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

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

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

2.6. Data Model Trees for Event Notifications

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

2.6.1. 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
```

2.6.2. 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
```

2.6.3. Data Model for RFC5277-bis Extensions

```
module: ietf-event-notifications
   +--ro streams
   |  +--ro stream*   notif:stream
   +--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?             stream
   |     +--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 subscription* [subscription-id]
         +--ro subscription-id            subscription-id
         +--ro configured-subscription?   empty {configured-subscriptions}?
         +--ro subscription-status?       subscription-status
         +--ro stream?                    stream
         +--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:
   +---- subscription-id?   subscription-id
augment /netmod-notif:notificationComplete:
   +---- subscription-id?   subscription-id
augment /netmod-notif:netconf/netmod-notif:streams:
   +--rw exclude-from-NETCONF-stream?   empty
rpcs:
   +---x establish-subscription
   |  +---w input
   |  |  +---w stream?       stream
   |  |  +---w (filter-type)?
   |  |  |  +--:(rfc5277)
   |  |  |  |  +---w filter
   |  |  |  +--:(by-reference)
   |  |  |     +---w filter-ref?   filter-ref
   |  |  +---w startTime?    yang:date-and-time
   |  |  +---w stopTime?     yang:date-and-time
   |  |  +---w encoding?     encoding
   |  +--ro output
   |     +--ro subscription-result    subscription-result
   |     +--ro (result)?
   |        +--:(success)
   |        |  +--ro subscription-id        subscription-id
   |        +--:(no-success)
   |           +--ro stream?                stream
   |           +--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
   |  +---w input
   |  |  +---w subscription-id?   subscription-id
   |  |  +---w stream?            stream
```

```
   |  |  +---w (filter-type)?
   |  |  |  +--:(rfc5277)
   |  |  |  |  +---w filter
   |  |  |  +--:(by-reference)
   |  |  |     +---w filter-ref?        filter-ref
   |  |  +---w startTime?         yang:date-and-time
   |  |  +---w stopTime?          yang:date-and-time
   |  |  +---w encoding?          encoding
   |  +--ro output
   |     +--ro subscription-result    subscription-result
   |     +--ro (result)?
   |        +--:(success)
   |        |  +--ro subscription-id        subscription-id
   |        +--:(no-success)
   |           +--ro stream?               stream
   |           +--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
      +---w input
      |  +---w subscription-id    subscription-id
      +--ro output
         +--ro subscription-result    subscription-result
notifications:
   +---n subscription-started
   |  +--ro subscription-id    subscription-id
   |  +--ro stream?            stream
   |  +--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?            stream
```

```
   |  +--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?             stream
   |  +--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
```

2.7. Creating a Subscription

Editor's note: The following section needs updating. NETCONF mapping will move to a separate document.

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 2.8

2.7.1. 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.

- o 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.

- o 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.

- o 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.

If the server can satisfy the request, it sends a positive acknowledgement.

If the request cannot be completed for any reason, an error is returned along with an error reason. 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. Other errors include:

- o The optional replay feature is requested but the server does not support it

- o A stopTime is requested that is earlier than the specified startTime

- o A startTime is requested that is later than the current time

2.8. Establishing a Subscription

Editor’s note: The following section needs updating. NETCONF mapping will move to a separate document.

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.

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

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

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

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. 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 is no guarantee that subsequent requests with those parameters for this client or others will be accepted. For instance, consider a subscription from [I-D.ietf-netconf-yang-push], which augments the establish-subscription with some additional parameters, including "period".

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

3. Modifying a Subscription

Editor's note: The following section needs updating. NETCONF mapping will move to a separate document.

This operation permits modifying the terms of a subscription previously established. Subscriptions created by configuration 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.

Configured 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) configured subscriptions via RPC.

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

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

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

4. Deleting a Subscription

Editor's note: The following section needs updating. NETCONF mapping will move to a separate document.

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. Configured 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 configured subscriptions.

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

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

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

5. Configured Subscriptions

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

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

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

In addition to subscription parameters that apply to dynamic subscriptions, the following additional parameters apply to configured subscriptions:

- 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.

5.1. Creating a Configured Subscription

Configured subscriptions cannot be created via configuration operations. New clients should use the mechanisms described in Section 5.2 for establishing configured subscriptions.

5.2. Establishing a Configured 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, configured 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 2: Establish configured 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 3: Response to a successful configured 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 4: Response to a failed configured subscription establishment

5.3. Modifying a Configured Subscription

Configured 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)

5.4. Deleting a Configured Subscription

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

```
<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 5: Deleting a configured 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).

6. 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 configured 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 6: 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 7: 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 8: Data plane notification using JSON encoding

7. 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 are unlike other notifications in that they are not general-purpose notifications. They cannot be filtered out, and they are delivered only to the receiver of a subscription. They are thus not part of the regular NETCONF event stream. The definition of control plane notifications is distinct from other notifications by making use of a YANG extension tagging them as control plane notification.

Control plane notifications include indications that a replay of notifications has been completed, that a subscription is done sending notifications because an end time has been reached, and that a subscription has started, been modified, been terminated, or been suspended. They are described in the following subsections.

7.1. 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.

7.2. 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.

7.3. subscription-started

This notification indicates that a configured 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.

7.4. subscription-modified

This notification indicates that a configured 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.

7.5. 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 configured subscriptions using configuration operations.

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

7.6. subscription-suspended

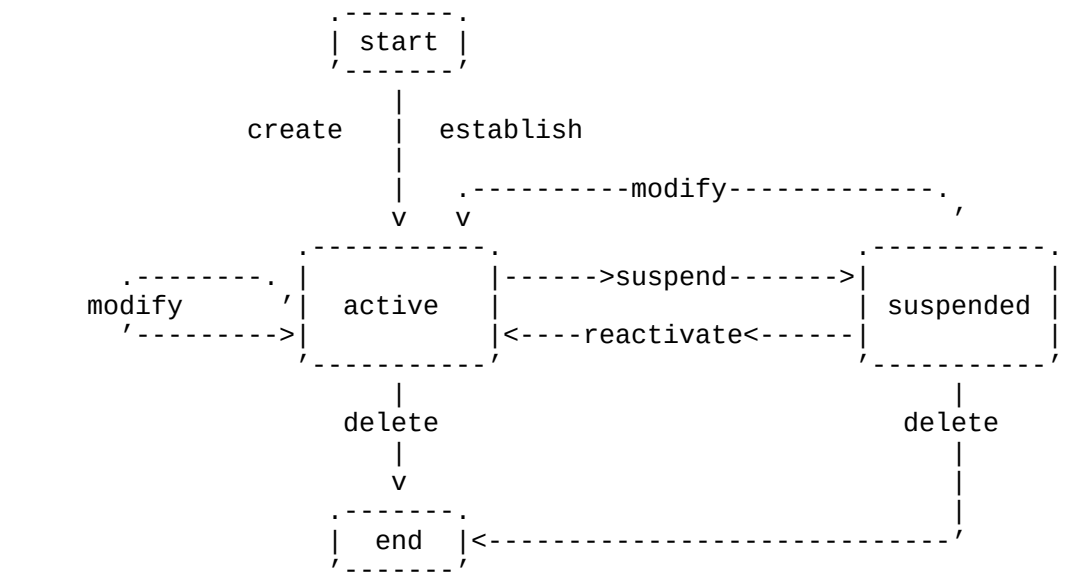
This notification indicates that a server has suspended a subscription. The notification includes the reason for the suspension. A possible reason is the lack of resources to serve it. No further data plane notifications will be sent until the subscription resumes. Suspensions are notified to the subscriber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

7.7. 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 configured subscriptions).

8. 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.



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.

9. Data Models for Event Notifications

9.1. Data Model for RFC5277 (netmod namespace)

```
<CODE BEGINS>
file "ietf-5277-netmod@2016-06-15.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 2016-06-15 {
```

```
    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>
```

9.2. Data Model for RFC5277 (netconf namespace)

```
  <CODE BEGINS>
  file "ietf-5277-netconf@2016-06-15.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 2016-06-15 {
       description
         "Model for RPC in RFC 5277: NETCONF Event Notifications";
       reference
         "RFC 5277: NETCONF Event Notifications";
     }

     /*
      * IDENTITIES
      */

     identity stream {
       description
         "Base identity to represent a generic stream of event
```

```
         notifications.";
    }

    identity NETCONF {
      base stream;
      description
        "Default NETCONF event stream, containing events based on
         notifications defined as YANG modules that are supported
         by the system.";
    }

    /*
     * TYPEDEFS
     */

    typedef stream {
      type identityref {
        base stream;
      }
      description
        "Specifies a system-provided datastream.";
    }

    /*
     * 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 stream;
        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>
```

9.3. Data Model for RFC5277-bis Extensions

```
  <CODE BEGINS>
  file "ietf-event-notifications@2016-06-15.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 2016-06-15 {
      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, deletion) 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-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).";
    }

    /* 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 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 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.";
    }

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

    /* 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 subscription-status {
      type identityref {
        base subscription-stream-status;
      }
      description
        "Specifies the status of a subscription or datastream.";
    }

    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 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 {
         uses subscription-response;
       }
     }

     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;
           mandatory true;
```

```
          description
            "Identifier of the subscription that is to be deleted.
             Only subscriptions that were created using
             create-subscription can be deleted via this RPC.";
        }
      }
      output {
        leaf subscription-result {
          type subscription-result;
          mandatory true;
          description
            "Indicates whether subscription is operational,
             or if a problem was encountered.";
        }
      }
    }


    /*
     * 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 streams {
      config false;
      description
        "This container contains a leaf list of built-in
         streams that are provided by the system.";
      leaf-list stream {
        type notif:stream;
        description
          "Identifies the built-in streams that are supported by the
           system.  Built-in streams are associated with their own
           identities, each of which carries a special semantics.
           In case configurable custom streams are supported,
           as indicated by the custom-stream identity, the configuration
           of those custom streams is provided separately.";
      }
    }
    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 subscription {
         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.";
         }

         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 subscription-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>
```

10. 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 9: 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.

11. 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 [I-D.ietf-netconf-yang-push], each of the elements in its data plane notifications must also go through access control.

12. Issues that are currently being worked and resolved

12.1. Unresolved and yet-to-be addressed issues

EN1 - Definition of basic set of Stream types. What streams are provided and what do they contain (includes default 5277 stream).

EN2 - Clarify interplay between filter definitions and different streams. Includes information in subtrees of event payloads.

EN3 - Mechanisms for diagnostics, e.g. deal with dropped updates, monitoring when they occur, etc

EN4 - How to allow for seamless integration with non-standard encodings and transports (like GPB/GRPC). Specify requirements encoding and transport must meet, provide examples.

EN5 - Along with Netconf-notif, should this draft obsolete 5277 or be in parallel with it?

EN6 - Stream discovery. Are adjustments needed for maximal transport independence?

EN7 - Detecting loss of a sequential update notification, and mechanisms to resend. Implications to transports must be thought through.

EN8 - Should we have a mandatory transport?

12.2. Agreement in principal

EN9 - Multiple receivers per Configured Subscription is ok.

EN10 - Replay support will be provided for selected stream types (modify vs. delete)

EN11 - Required layering security requirements/considerations will be added into the YANG model for Configured Subscriptions. It will be up to the transport to meet these requirements.

EN12 - Test-only option for a subscription is desired. But it still needs to be defined.

EN13 - RFC6241 Subtree-filter definition in 5277bis cannot apply to elements of an event. Must explicitly define how 6241 doesn't apply filtering within a 5277bis event.

EN14 - Ensure that Configured Subscriptions are fully defined in YANG model.

12.3. Resolved Issues

EN15 - Term for Dynamic and Static Subscriptions (move to "Configured")

12.4. Editorial To-Dos

Update examples. Examples are not in synch with the current model.

Remove overlap with transport specifics, specifically NETCONF. The current draft revision is "self-contained" and assumes NETCONF as a transport. It does not yet reflect the breakout of transport mappings into a separate draft.

13. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Yang Geng, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, and Guangying Zheng.

14. References

14.1. 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>.

14.2. Informative References

[I-D.ietf-netconf-restconf] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-13, April 2016.

[I-D.ietf-netconf-yang-push] Clemm, Alexander., Gonzalez Prieto, Alberto., Voit, Eric., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", June 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

Sharon Chisholm
Ciena

Email: schishol@ciena.com

Hector Trevino
Cisco Systems

Email: htrevino@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2016

A. Gonzalez Prieto
A. Clemm
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
S. Chisholm
Ciena
H. Trevino
Cisco Systems
June 15, 2016

NETCONF Support for Event Notifications
draft-gonzalez-netconf-event-notifications-00

Abstract

This document defines the support of [event-notifications] by the Network Configuration protocol (NETCONF). [event-notifications] describes capabilities and operations for providing asynchronous message notification delivery. This document discussed how to provide them on top of NETCONF. The capabilities and operations defined between this document and [event-notifications] are intended to obsolete RFC 5277.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

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

Copyright Notice

Table of Contents

1. Introduction

[RFC6241] can be conceptually partitioned into four layers:

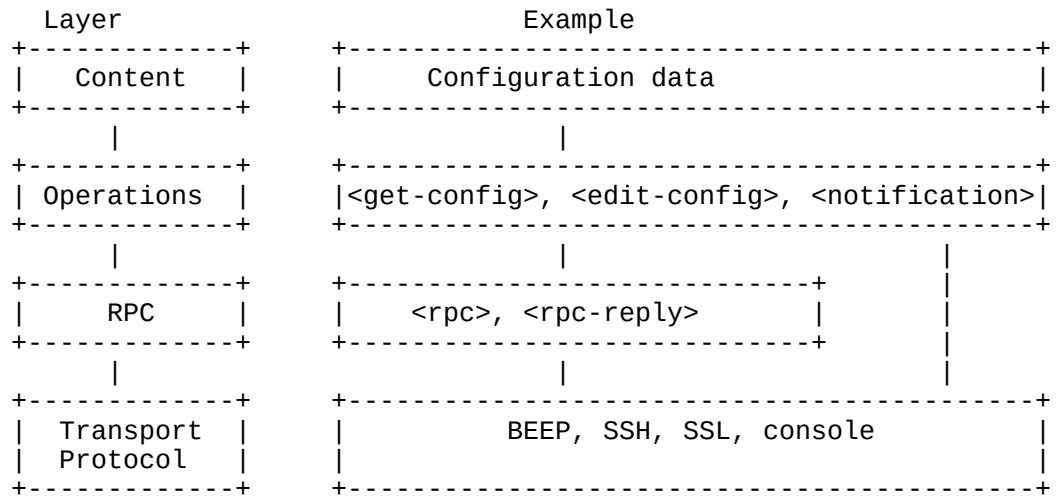


Figure 1: NETCONF layer architecture

This document defines mechanisms that provide an asynchronous message notification delivery service for the NETCONF protocol [RFC6241] based on [event-notifications]. This is an optional capability built on top of the base NETCONF definition.

[event-notifications] and this document enhance the capabilities of RFC 5277 while maintaining backwards capability with existing implementations. It is intended that a final version of this document might obsolete [RFC5277].

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.

[event-notifications] covers the following functionality:

- Ability to subscribe to event notifications using two mechanisms: dynamic and configuration subscriptions.

- Ability to subscribe to event notifications using two mechanisms: dynamic and configuration subscriptions.

- Ability to negotiate acceptable subscription parameters.

- Ability to filter the subset of notifications to be pushed with stream-specific semantics.
- Ability to support multiple encodings for the notification.
- Mechanism to communicate the notifications.
- Ability to replay locally logged notifications.

To support this functionality, NETCONF agents must implement the operations, configuration and operational state defined in [event-notifications]. In addition, they need to:

- support multiple subscriptions over a single NETCONF session.
- support a revised definition of the default NETCONF stream
- be backwards compatible with RFC 5277 implementations.

1.1. 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].

1.1.1. Event Notifications

The following terms are defined in [event-notifications]:

- Event
- Event notification
- Stream (also referred to as "event stream")
- Subscriber
- Receiver
- Subscription
- Filter
- Dynamic subscription
- Configured subscription

1.1.2. NETCONF

The following terms are defined in [RFC6241] :

- Operation
- RPC: remote procedure call

1.1.3. NETCONF Access Control

The following terms are defined in [RFC6536] :

- NACM: NETCONF Access Control Model

1.2. Solution Overview

[event-notifications] defines mechanisms that provide an asynchronous message notification delivery service. This document discusses its realization on top of the NETCONF protocol [RFC6241].

The functionality to support is defined in [event-notifications]. It is formalized in a set of yang models. The mapping of yang constructs into NETCONF is described in [RFC6020].

Supporting [event-notifications] requires enhancements and modifications in NETCONF. The key enhacement is suporting multiple subscriptions on a NETCONF session. A key modification is the definition of the NETCONF stream

These enhancements do not affect [RFC5277] clients that do not support [event-notifications].

2. Solution

In this section, we describe and exemplify how [event-notifications] must be supported over NETCONF.

2.1. Event Streams

In the context of NETCONF, 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 "/streams", which includes the stream identifier.

The following example ilustrates the retrieval of the list of available event streams using the <get> operation.

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

Figure 2: 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>
    <streams
        xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications">
      <stream>NETCONF</stream>
      <stream>SNMP</stream>
      <stream>syslog-critical</stream>
      <stream>NETCONF</stream>
      </streams>
  </data>
</rpc-reply>
```

Figure 3: Get streams response

2.2.1. Backwards Compatibility

In order to maintain backwards compatibility, clients that only support [RFC5277] can retrieve the list of available event streams executing a <get> operation against the container "/netconf/streams".

The following example ilustrates this mechanism.

```
<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 4: Get streams (backwards compatibility)

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 5: Get streams response (backwards compatibility)

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. Creating a Subscription

The following illustrates the creation of a simple subscription.

2.4.1. Usage Example

The following demonstrates dynamically creating a subscription. This operation is fully defined in [RFC5277]

```
<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 6: Create subscription

2.4.2. 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 7: Successful create subscription

2.4.3. 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.
```

Figure 8: Create subscription missing an element

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.
```

Figure 9: Create subscription pperation failed

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.
```

Figure 10: Create subscription incorrect stopTime

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.
```

Figure 11: Create subscription incorrect startTime

2.5. Establishing a Subscription

2.5.1. Usage Example

The following illustrates the establishment of 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 12: Establish subscription

2.5.2. 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 13: Successful establish subscription

2.5.3. 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 14: 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 15: 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.

2.5.4. Multiple Subscriptions over a Single NETCONF Session

Note that [event-notifications] requires supporting multiple subscriptions over a single NETCONF session. In contrast, [RFC5277] mandated server to return an error when a create-subscription was sent while a subscription was active on that session.

2.5.5. Message Flow Examples

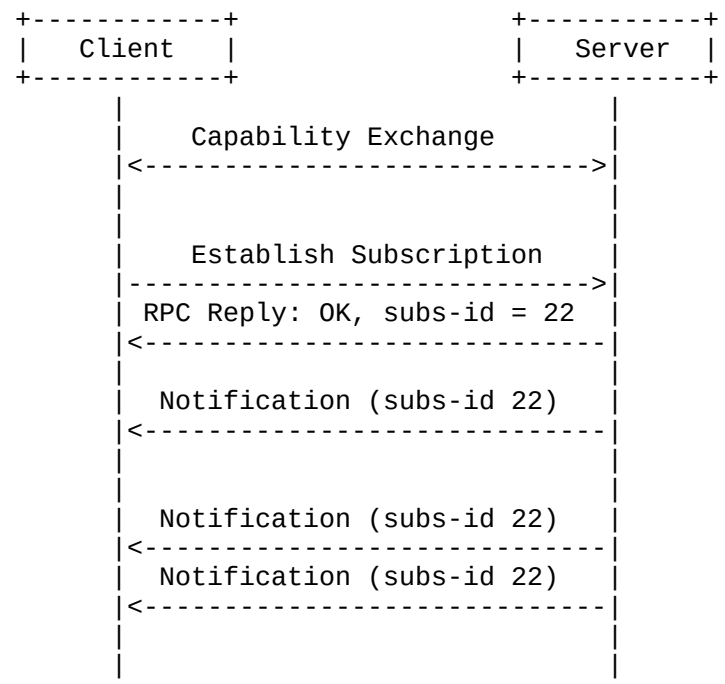


Figure 16: Message flow for subscription establishment

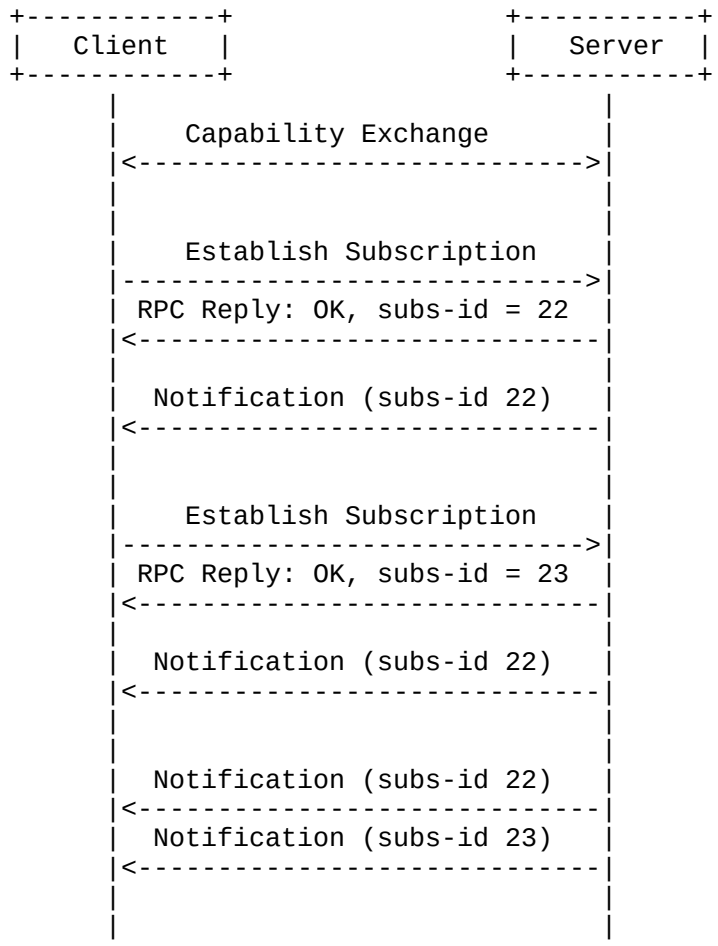


Figure 17: Message Flow for multiple subscription establishments over a single session

2.6. Modifying a Subscription

2.6.1. 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 18: 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 19: Modify subscription

2.6.2. 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 20: Successful modify subscription

2.6.3. 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 21: Unsuccessful modify subscription

2.6.4. Message Flow Example

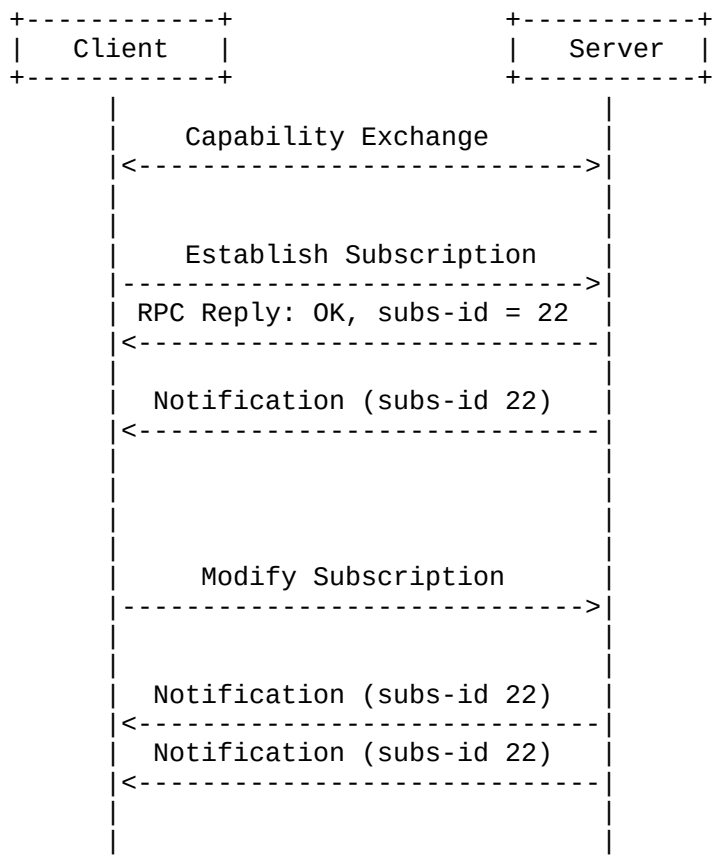


Figure 22: Message flow for subscription modification

2.7. Deleting a Subscription

2.7.1. 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 23: Delete subscription

2.7.2. 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 24: Successful delete subscription

2.7.3. 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 25: Unsuccessful delete subscription

2.7.4. Message Flow Example

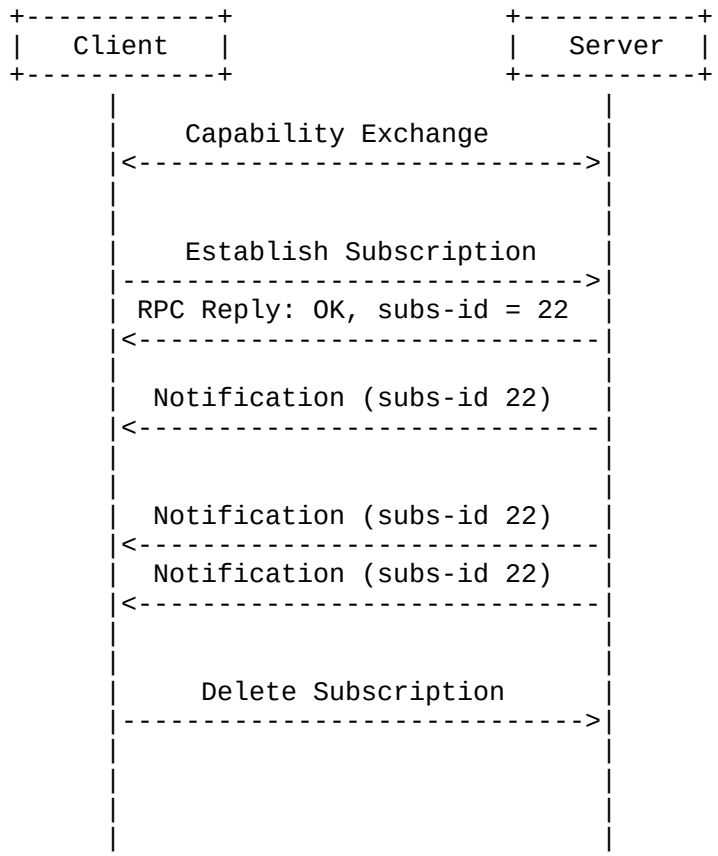


Figure 26: Message flow for subscription deletion

2.8. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface, and do not support negotiation.

Supporting configured subscriptions is optional and advertised during the capabilities exchange using the "configured-subscriptions" feature.

In this section, we present examples of how to manage configuration subscriptions using a NETCONF client.

2.8.1. Establishing a Configured Subscription

Subscriptions are established using configuration operations against the top-level subtree subscription-config.

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 27: 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 28: 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 29: Response to a failed static subscription establishment

2.8.1.1. Message Flow Example

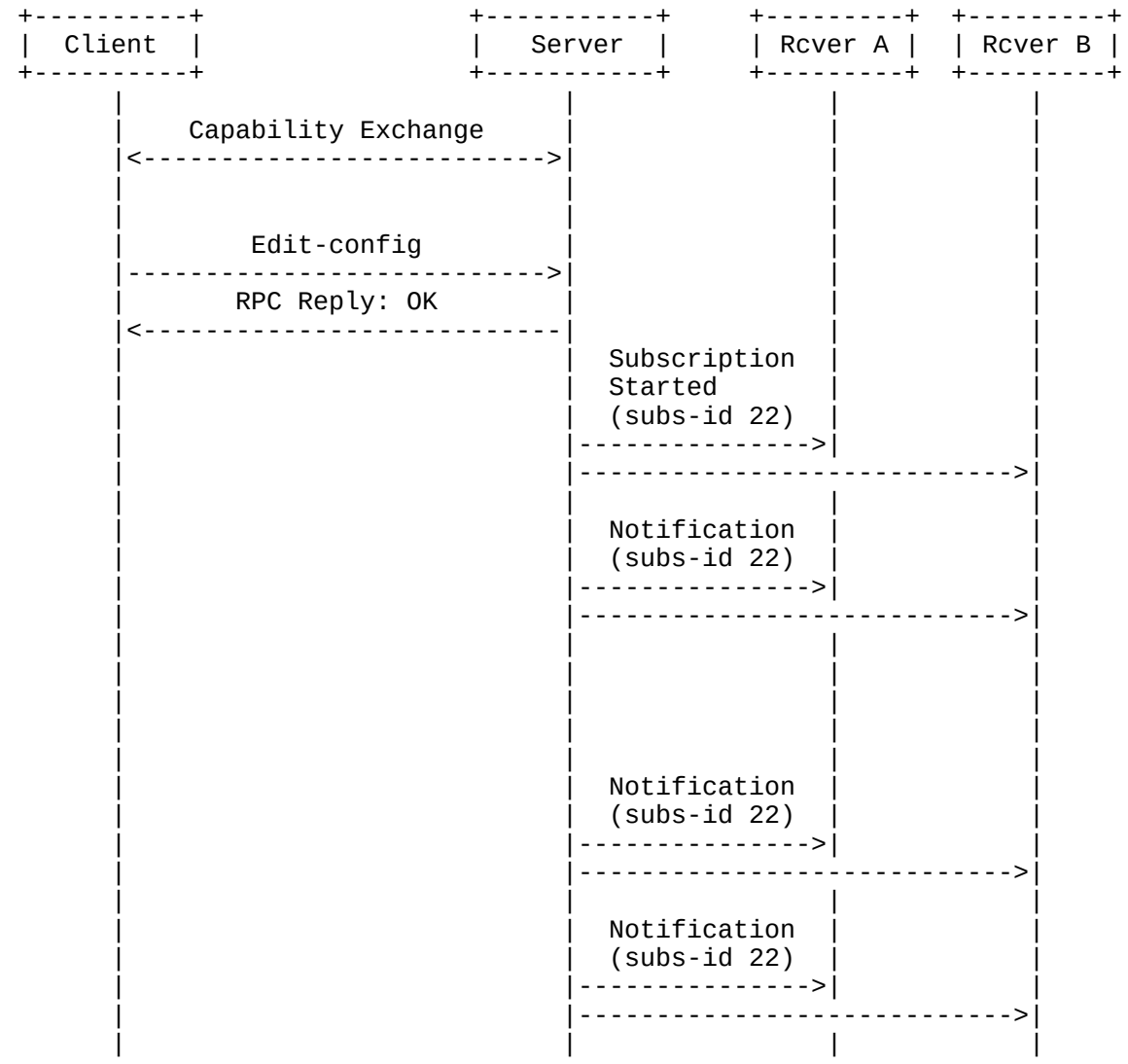


Figure 30: Message flow for subscription establishment (configured subscription)

2.8.2. Modifying a Configured Subscription

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

For example, the subscription established in the previous section could be modified as follows, choosing a different receiver:

```
<rpc message-id="102"
       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>
            1.2.3.5
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 31: Modify configured subscription

if the request is accepted, the server would reply:

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

Figure 32: Response to a successful configured subscription modification

2.8.2.1. Message Flow Example

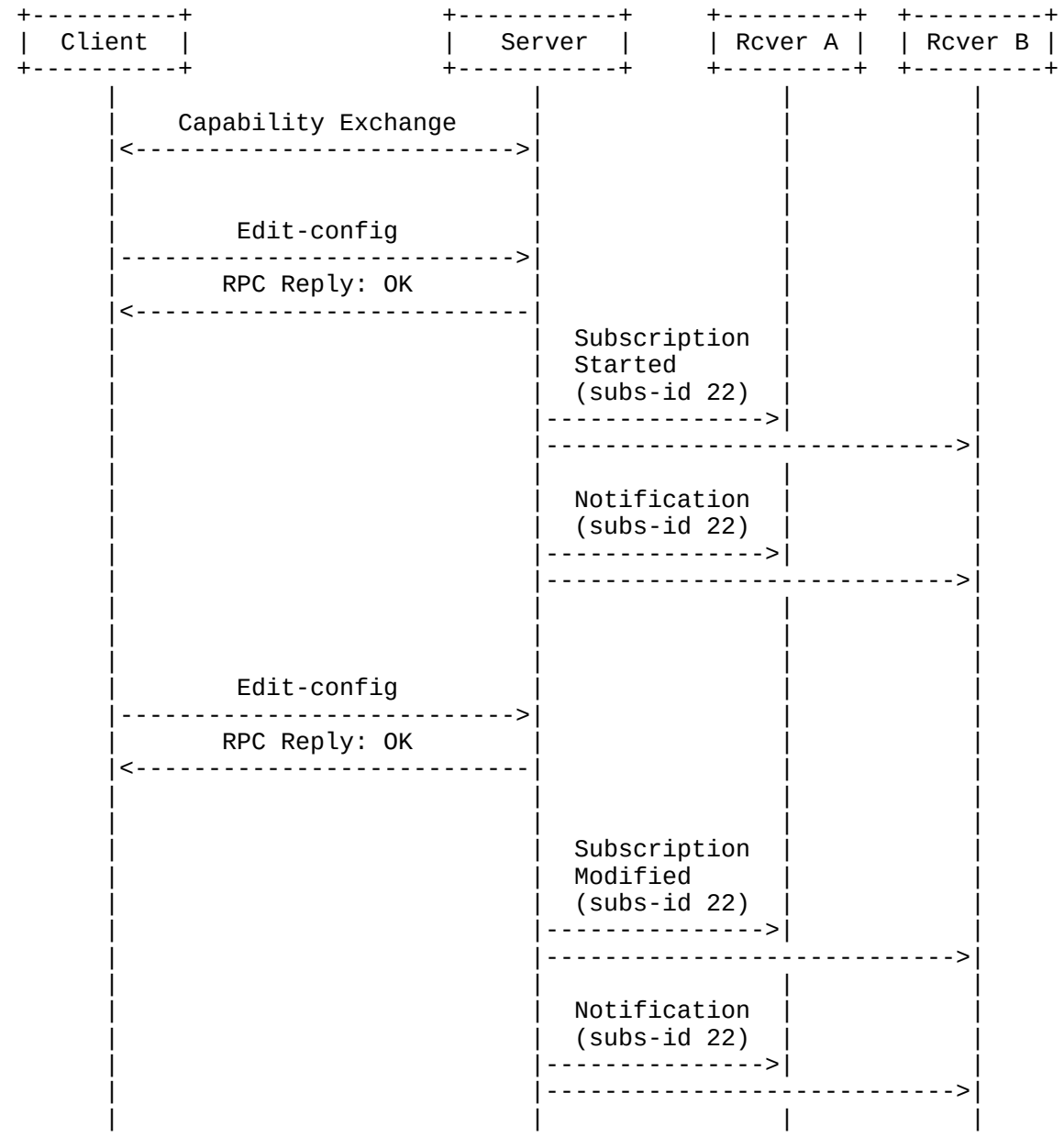


Figure 33: Message flow for subscription modification (configured subscription)

2.8.3. Deleting a Configured Subscription

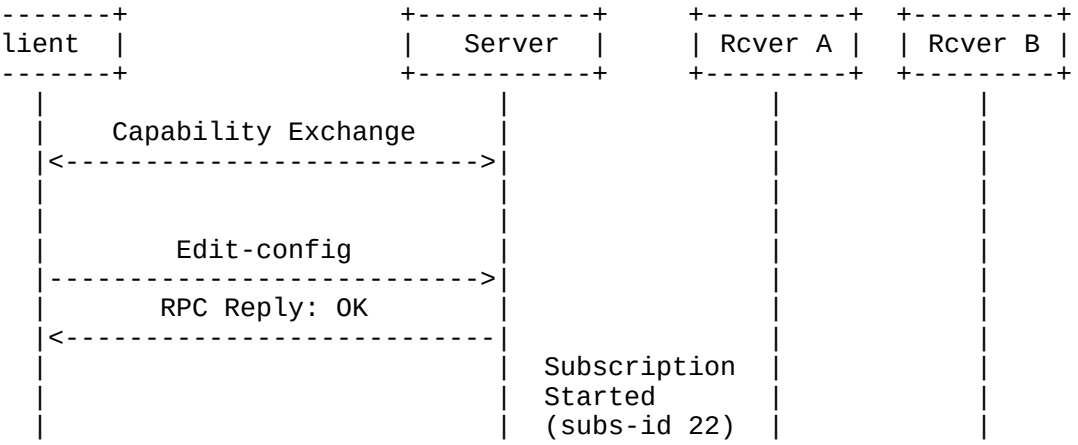
Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example:

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

Figure 34: Deleting a configured subscription

2.8.3.1. Message Flow Example



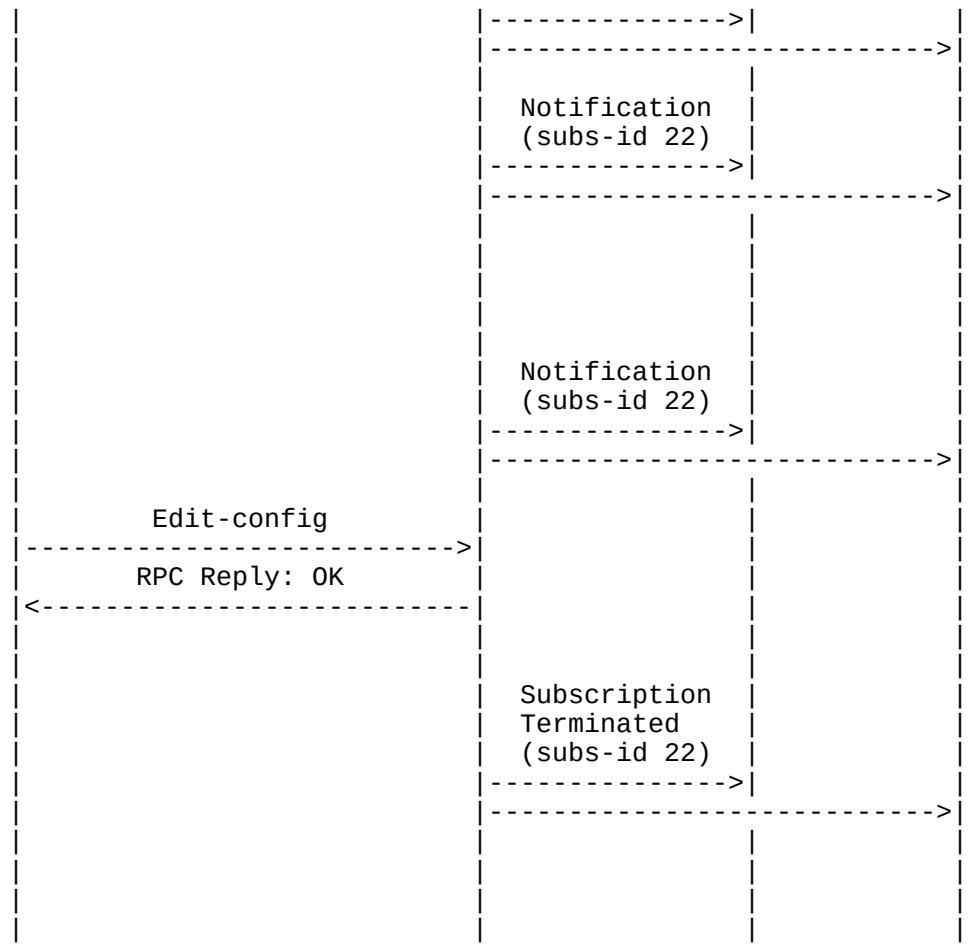


Figure 35: Message flow for subscription deletion (configured subscription)

2.9. 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 36: 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 37: 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 38: Data plane notification using JSON encoding

2.10. Control Plane Notifications

In addition to data plane notifications, a server may send control plane notifications (defined in [event-notifications]) to indicate to receivers that an event related to the subscription management has occurred. Control plane notifications cannot be filtered out. Next we exemplify them using both XML, and JSON encondings for the data:

2.10.1. replayComplete

```
<notification
       xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <replayComplete xmlns="urn:ietf:params:xml:ns:netmod:notification"/>
</notification>
```

Figure 39: replayComplete control 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>
       {
          "netmod-notif:replayComplete": { }
        }
  </notification-contents-json>
</notification>
```

Figure 40: replayComplete control plane notification (JSON encoding)

2.10.1.1. Message Flow Example

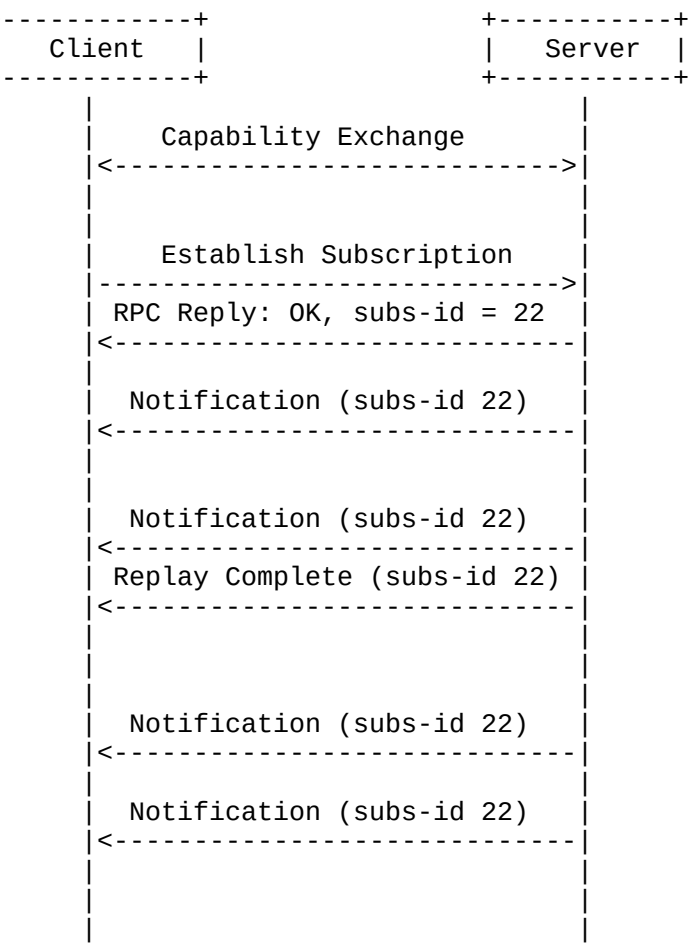


Figure 41: replayComplete notification

2.10.2. notificationComplete

```
<notification
    xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notificationComplete
    xmlns="urn:ietf:params:xml:ns:netmod:notification"/>
</notification>
```

Figure 42: notificationComplete control plane notification

2.10.2.1. Message Flow Example

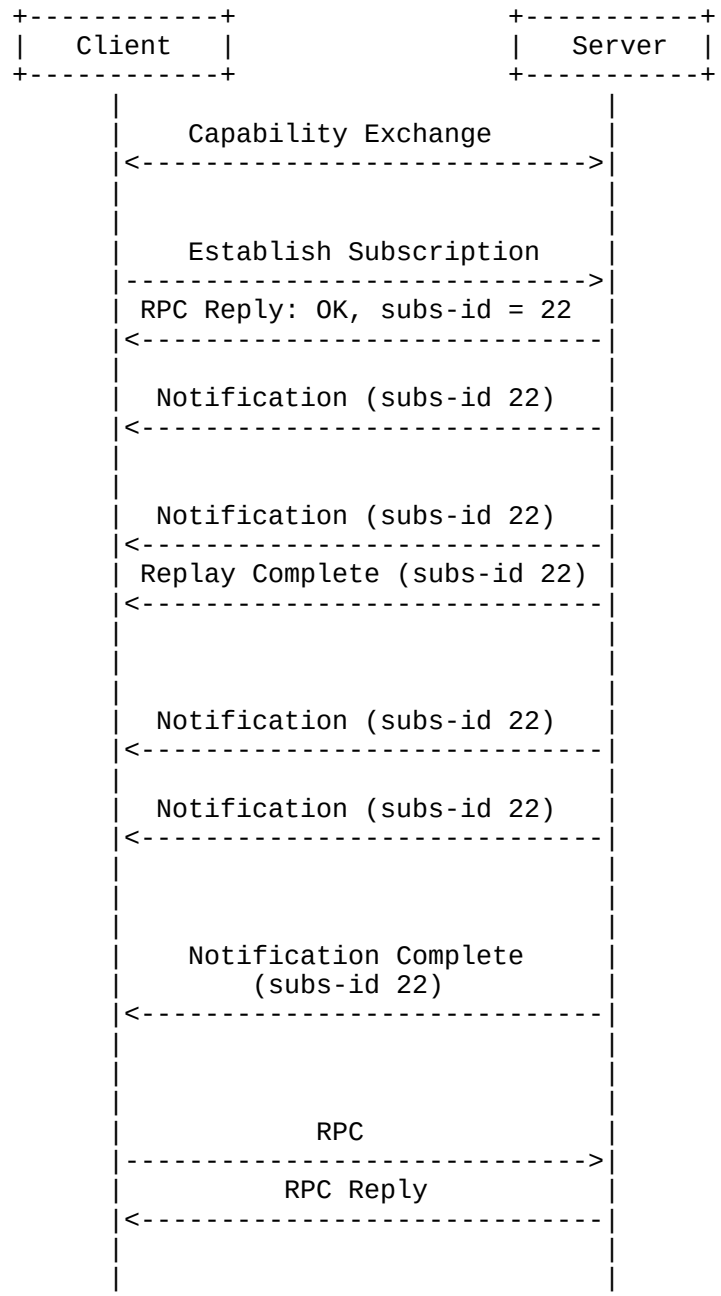


Figure 43: notificationComplete notification

2.10.3. subscription-started

```
  <notification
      xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-started
          xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
      <subscription-id>52</subscription-id>
      <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')]"/>
    </subscription-started/>
  </notification>
```

Figure 44: subscription-started control 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>
       {
          "notif-bis:subscription-started": {
              "subscription-id" : 52
              ((Open Item: express filter in json))
          }
        }
  </notification-contents-json>
</notification>
```

Figure 45: subscription-started control plane notification (JSON encoding)

2.10.4. subscription-modified

```
  <notification
      xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-modified
          xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
      <subscription-id>52</subscription-id>
      <filter netconf:type="xpath"
                xmlns:ex="http://example.com/event/1.0"
                select="/ex:event[ex:eventClass='fault']"/>
    </subscription-modified/>
  </notification>
```

Figure 46: subscription-modified control plane notification

2.10.5. subscription-terminated

```
  <notification
      xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-terminated
          xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
      <subscription-id>52</subscription-id>
      <reason>subscription-deleted</reason>
    </subscription-terminated/>
  </notification>
```

Figure 47: subscription-terminated control plane notification

2.10.6. subscription-suspended

```
  <notification
      xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-suspended
          xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
      <subscription-id>52</subscription-id>
      <reason>internal-error</reason>
    </subscription-suspended/>
  </notification>
```

Figure 48: subscription-suspended control plane notification

2.10.7. subscription-resumed

```
  <notification
      xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-resumed
          xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications"/>
      <subscription-id>52</subscription-id>
      <reason>internal-error</reason>
    </subscription-resumed/>
  </notification>
```

Figure 49: subscription-resumed control plane notification

2.10.7.1. Message Flow Examples

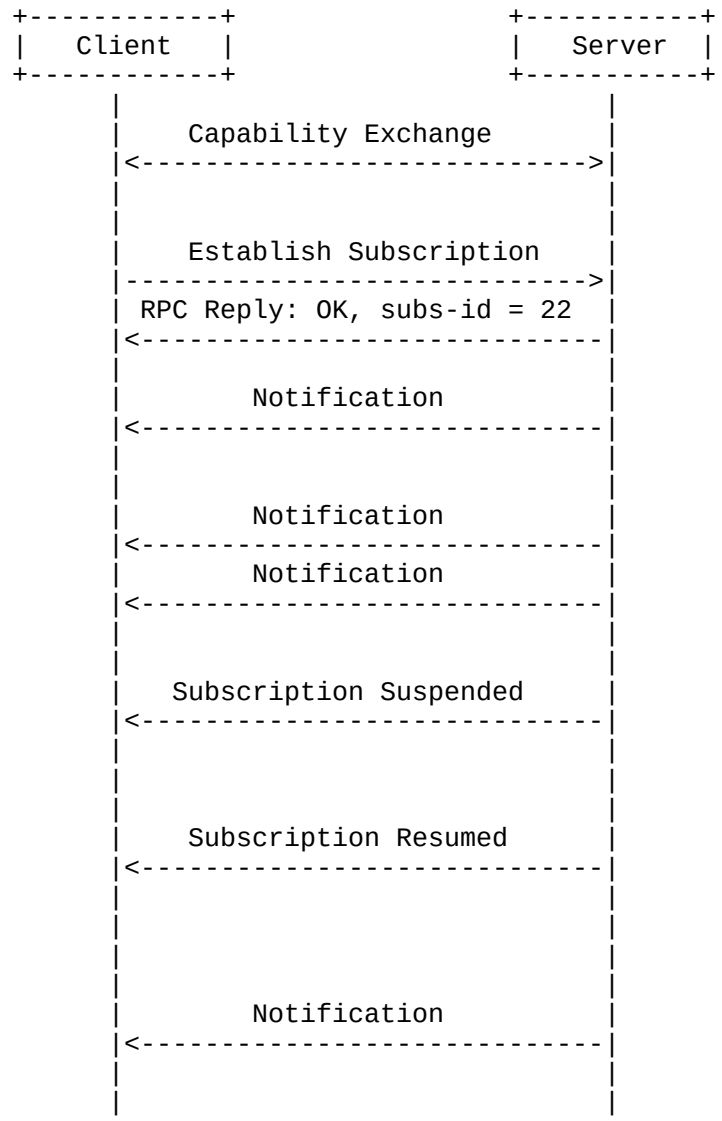
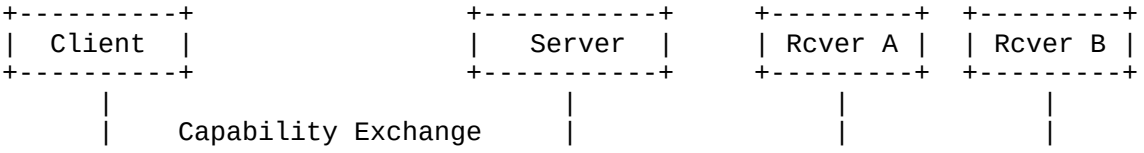


Figure 50: subscription-suspended and Resumed Notifications



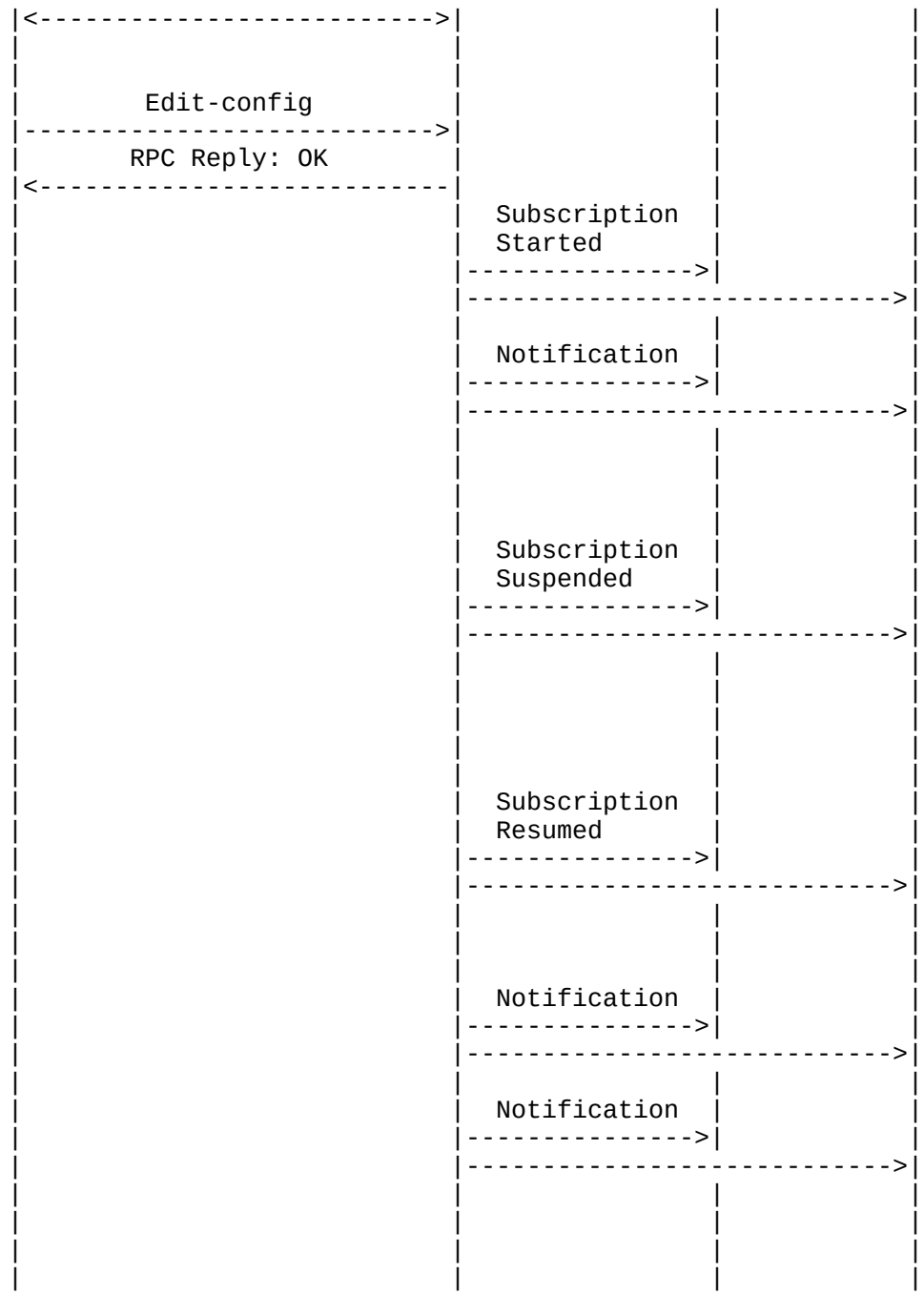

Edit-config
RPC Reply: OK
Subscription Started
Notification
Subscription Suspended
Subscription Resumed
Notification
Notification

Figure 51: subscription-suspended and subscription-resumed notifications (configured subscriptions)

3. Backwards Compatibility

3.1. Capabilities

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 52: 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.

3.2. Stream Discovery

In order to maintain backwards compatibility, clients that only support [RFC5277] can retrieve the list of available event streams executing a <get> operation against the container "/netconf/streams".

4. 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.

5. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Andy Bierman, Yan Gang, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, and Guangying Zheng.

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

[event-notifications]
Gonzalez Prieto, A., Clemm, A., Voit, Eric., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "Subscribing to YANG-Defined Event Notifications", June 2016, <https://datatracker.ietf.org/doc/draft-gonzalez-netconf-5277bis/>.

[netconf-yang-push]
Clemm, A., Gonzalez Prieto, A., 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/>.

Appendix A. Issues being worked and resolved

(To be removed by RFC editor prior to publication)

A.1. Unresolved Issues

NT1 - Support multiple create-subscriptions over a single NETCONF session? or only multiple establish-subscription? Recommend only establish-subscription anytime there may be more than one subscription, otherwise variations of supported RFCs and error states proliferate.

NT2 - Configured subscription need to be refined in [event-notifications] and then adjust this document based on it.

NT3 - Express filter in JSON should be documented.

A.2. Agreement in principal

NT1 - When a list of requirements for secure transport is provided via a Configured Subscription, this list must be met via Netconf transport. Specifics still are needed for draft on the "how".

NT2 - Along with rfc5277bis, this draft to obsolete 5277

NT3 - Stream discovery. Adopted mechanism in 5277-bis and include backwards comptability support.

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

Sharon Chisholm
Ciena

Email: schishol@ciena.com

Hector Trevino
Cisco Systems

Email: htrevino@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Networks
J. Schoenwaelder
Jacobs University Bremen
July 8, 2016

NETCONF Client and Server Models
draft-ietf-netconf-netconf-client-server-00

Abstract

This document defines two YANG modules, one module to configure a NETCONF client and the other module to configure a NETCONF server. Both modules support both the SSH and TLS transport protocols, and support both standard NETCONF and NETCONF Call Home connections.

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.

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-system-keychain
- draft-ietf-netconf-ssh-client-server
- draft-ietf-netconf-tls-client-server

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-ssh-client-server
- "ZZZZ" --> the assigned RFC value for draft-ietf-netconf-tls-client-server

o "AAAA" --> the assigned RFC value for draft-ietf-netconf-call-home

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

o "2016-07-08" --> the publication date of this draft

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

o Appendix A. Change Log

o Appendix B. Open Issues

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines two YANG [RFC6020] modules, one module to configure a NETCONF client and the other module to configure a NETCONF server. Both modules support both the SSH and TLS transport protocols, and support both standard NETCONF and NETCONF Call Home connections.

NETCONF is defined by [RFC6241]. SSH is defined by [RFC4252], [RFC4253], and [RFC4254]. TLS is defined by [RFC5246]. NETCONF Call Home is defined by [draft-ietf-netconf-call-home]).

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. The NETCONF Client Model

The NETCONF client model presented in this section supports both clients initiating connections to servers, as well as clients listening for connections from servers calling home.

This model supports both the SSH and TLS transport protocols, using the SSH client and TLS client groupings defined in [draft-ietf-netconf-ssh-client-server] and [draft-ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keychain model defined in [draft-ietf-netconf-system-keychain].

YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF client supports.

2.1. Tree Diagram

Note: all lines are folded at column 71 with no '\' character.

```
module: ietf-netconf-client
   +--rw netconf-client
      +--rw initiate {initiate}?
      |  +--rw netconf-server* [name]
      |     +--rw name    string
      |     +--rw (transport)
      |        +--:(ssh) {ssh-initiate}?
      |           +--rw ssh
      |              +--rw address        inet:host
      |              +--rw port?          inet:port-number
      |              +--rw server-auth
      |              |  +--rw trusted-ssh-host-keys?   -> /kc:keychain/
trusted-ssh-host-keys/name
      |              |  +--rw trusted-ca-certs?        -> /kc:keychain/
trusted-certificates/name {ssh-x509-certs}?
      |              |  +--rw trusted-server-certs?    -> /kc:keychain/
trusted-certificates/name
      |              +--rw client-auth
      |                 +--rw matches* [name]
      |                    +--rw name                      string
      |                    +--rw match* [name]
      |                    |  +--rw name                      string
      |                    |  +--rw trusted-ssh-host-keys?   -> /kc:key
chain/trusted-ssh-host-keys/name
      |                    |  +--rw trusted-ca-certs?        -> /kc:key
chain/trusted-certificates/name
      |                    |  +--rw trusted-server-certs?    -> /kc:key
chain/trusted-certificates/name
      |                    +--rw user-auth-credentials?   -> /kc:keycha
in/user-auth-credentials/user-auth-credential/username
      +--rw listen {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 server-auth
```

```
                       |  +--rw trusted-ssh-host-keys?   -> /kc:keychain/
 trusted-ssh-host-keys/name
                       |  +--rw trusted-ca-certs?        -> /kc:keychain/
 trusted-certificates/name {ssh-x509-certs}?
                       |  +--rw trusted-server-certs?    -> /kc:keychain/
 trusted-certificates/name
                       +--rw client-auth
                          +--rw matches* [name]
                             +--rw name                       string
                             +--rw match* [name]
                             |  +--rw name                       string
                             |  +--rw trusted-ssh-host-keys?   -> /kc:key
 chain/trusted-ssh-host-keys/name
                             |  +--rw trusted-ca-certs?        -> /kc:key
 chain/trusted-certificates/name
                             |  +--rw trusted-server-certs?    -> /kc:key
 chain/trusted-certificates/name
                             +--rw user-auth-credentials?   -> /kc:keycha
 in/user-auth-credentials/user-auth-credential/username
```

2.2. Example Usage

The following example illustrates configuring a NETCONF client to initiate connections, using both the SSH and TLS transport protocols, as well as listening for call-home connections, again using both the SSH and TLS transport protocols.

This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

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

  <!-- NETCONF servers to initiate NETCONF connections to -->
  <initiate>
    <netconf-server>
      <name>corp-fw1</name>
      <ssh>
        <address>corp-fw1.example.com</address>
        <server-auth>
          <trusted-server-certs>
            deployment-specific-ca-certs
          </trusted-server-certs>
        </server-auth>
        <client-auth>
          <matches>
            <match>
              <trusted-ca-certs>
```

```
                  deployment-specific-ca-certs
                </trusted-ca-certs>
              </match>
              <user-auth-credentials>Bob</user-auth-credentials>
            </matches>
          </client-auth>
        </ssh>
      </netconf-server>
    </initiate>

    <!-- endpoints to listen for NETCONF Call Home connections on -->
    <listen>
      <endpoint>
        <name>Intranet-facing listener</name>
        <ssh>
          <address>11.22.33.44</address>
          <server-auth>
            <trusted-ca-certs>
              deployment-specific-ca-certs
            </trusted-ca-certs>
            <trusted-server-certs>
              explicitly-trusted-server-certs
            </trusted-server-certs>
            <trusted-ssh-host-keys>
              explicitly-trusted-ssh-host-keys
            </trusted-ssh-host-keys>
          </server-auth>
          <client-auth>
            <matches>
              <match>
                <trusted-ca-certs>
                  deployment-specific-ca-certs
                </trusted-ca-certs>
              </match>
              <user-auth-credentials>admin</user-auth-credentials>
            </matches>
            <matches>
              <match>
                <trusted-ca-certs>
                  explicitly-trusted-server-certs
                </trusted-ca-certs>
              </match>
              <user-auth-credentials>admin</user-auth-credentials>
            </matches>
            <matches>
              <match>
                <trusted-ca-certs>
                  explicitly-trusted-ssh-host-keys
```

```
                </trusted-ca-certs>
              </match>
              <user-auth-credentials>admin</user-auth-credentials>
            </matches>
          </client-auth>
        </ssh>
      </endpoint>
    </listen>
  </netconf-client>
```

2.3. YANG Model

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

<CODE BEGINS> file "ietf-netconf-client@2016-07-08.yang"

```
module ietf-netconf-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-client";
  prefix "ncc";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: A YANG Data Model for SNMP Configuration";
  }

  import ietf-ssh-client {
    prefix ss;
    revision-date 2016-07-08; // stable grouping definitions
    reference
      "RFC YYYY: SSH Client and Server Models";
  }

  import ietf-tls-client {
    prefix ts;
    revision-date 2016-07-08; // stable grouping definitions
    reference
      "RFC ZZZZ: TLS Client and Server Models";
  }
```

```
  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>

    Author:   Kent Watsen
              <mailto:kwatsen@juniper.net>

    Author:   Gary Wu
              <mailto:garywu@cisco.com>";


  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 XXXX; see
    the RFC itself for full legal notices.";

  revision "2016-07-08" {
    description
     "Initial version";
    reference
     "RFC XXXX: NETCONF Client and Server Models";
  }

  // Features

  feature initiate {
    description
```

```
     "The 'initiate' feature indicates that the NETCONF client
      supports initiating NETCONF connections to NETCONF servers
      using at least one transport (e.g., SSH, TLS, etc.).";
  }

  feature ssh-initiate {
    description
     "The 'ssh-initiate' feature indicates that the NETCONF client
      supports initiating SSH connections to NETCONF servers.";
    reference
     "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
  }

  feature tls-initiate {
    description
     "The 'tls-initiate' feature indicates that the NETCONF client
      supports initiating TLS connections to NETCONF servers.";
    reference
     "RFC 7589: Using the NETCONF Protocol over Transport
                Layer Security (TLS) with Mutual X.509
                Authentication";
  }

  feature listen {
    description
     "The 'listen' feature indicates that the NETCONF client
      supports opening a port to accept NETCONF server call home
      connections using at least one transport (e.g., SSH, TLS, etc.).";
  }

  feature ssh-listen {
    description
     "The 'ssh-listen' feature indicates that the NETCONF client
      supports opening a port to listen for incoming NETCONF
      server call-home SSH connections.";
    reference
     "RFC AAAA: NETCONF Call Home and RESTCONF Call Home";
  }

  feature tls-listen {
    description
     "The 'tls-listen' feature indicates that the NETCONF client
      supports opening a port to listen for incoming NETCONF
      server call-home TLS connections.";
    reference
     "RFC AAAA: NETCONF Call Home and RESTCONF Call Home";
  }
```

```
  container netconf-client {
    description
      "Top-level container for NETCONF client configuration.";

    container initiate {
      if-feature initiate;
      description
        "Configures client intiating underlying TCP connections.";
      list netconf-server {
        key name;
        description
          "List of NETCONF servers the NETCONF client is to initiate
           connections to.";
        leaf name {
          type string;
          description
            "An arbitrary name for the NETCONF server.";
        }
        choice transport {
          mandatory true;
          description
            "Selects between available transports.";
          case ssh {
            if-feature ssh-initiate;
            container ssh {
              description
                "Specifies SSH-specific transport configuration.";
              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 client
                  will process the IP addresses as if they had been
                  explicitly configured in place of the hostname.";
              }
              leaf port {
                type inet:port-number;
                default 830;
                description
                 "The IP port for this endpoint. The NETCONF client will
                  use the IANA-assigned well-known port if no value is
                  specified.";
              }
              uses ss:initiating-ssh-client-grouping;
            }
          }
```

```
/*
          case tls {
            if-feature tls-initiate;
            container tls {
              description
                "Specifies TLS-specific transport configuration.";
              uses endpoints-container {
                refine endpoints/endpoint/port {
                  default 6513;
                }
              }
              uses ts:listening-tls-client-grouping {
                augment "client-auth" {
                  description
                    "Augments in the cert-to-name structure.";
                  uses cert-maps-grouping;
                }
              }
            }
          }
*/
        }
      }
    } // end initiate

    container listen {
      if-feature listen;
      description
        "Configures client accepting call-home TCP connections.";
      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-client-grouping {
                refine port {
                  default 4334;
                }
              }
            }
          }
/*
          case tls {
            if-feature tls-listen;
            container tls {
              description
                "TLS-specific listening configuration for inbound
                 connections.";
              uses ts:listening-tls-client-grouping {
                refine port {
                  default 4335;
                }
                augment "client-auth" {
                  description
                    "Augments in the cert-to-name structure.";
                  uses cert-maps-grouping;
                }
              }
            }
          }
*/
        }
```

```
      }
    } // end listen
  }

  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";
    }
  }

}
```

<CODE ENDS>

3. The NETCONF Server Model

The NETCONF server model presented in this section supports servers both listening for connections 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 defined in [draft-ietf-netconf-ssh-client-server] and [draft-ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keychain model defined in [draft-ietf-netconf-system-keychain].

YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF server supports.

3.1. Tree Diagram

Note: all lines are folded at column 71 with no '\' character.

```
module: ietf-netconf-server
   +--rw netconf-server
      +--rw session-options
      |  +--rw hello-timeout?   uint16
      +--rw listen {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/name {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/name
      |              +--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 {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/name {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/name
            |        +--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
```

3.2. Example Usage

The following example illustrates 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 the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

```
<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>
```

3.3. YANG Model

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

<CODE BEGINS> file "ietf-netconf-server@2016-07-08.yang"

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

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

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
      "RFC 7407: A YANG Data Model for SNMP Configuration";
  }

  import ietf-ssh-server {
    prefix ss;
    revision-date 2016-07-08; // stable grouping definitions
    reference
      "RFC YYYY: SSH Client and Server Models";
  }

  import ietf-tls-server {
    prefix ts;
    revision-date 2016-07-08; // stable grouping definitions
    reference
      "RFC ZZZZ: TLS Client and Server Models";
  }

  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 XXXX; see
      the RFC itself for full legal notices.";

    revision "2016-07-08" {
      description
       "Initial version";
      reference
       "RFC XXXX: NETCONF Client and Server Models";
    }


    // Features

    feature listen {
      description
       "The 'listen' feature indicates that the NETCONF server
        supports opening a port to accept NETCONF client connections
        using at least one transport (e.g., SSH, TLS, etc.).";
    }

    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 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 call-home {
      description
       "The 'call-home' feature indicates that the NETCONF server
        supports initiating NETCONF call home connections to NETCONF
        clients using at least one transport (e.g., SSH, TLS, etc.).";
      reference
       "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
    }

    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-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";
    }


    // 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 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 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 4334;
                 }
               }
               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 4335;
                 }
               }
               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 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 must close
                the connection when it is ready to release it. Once
                the connection has been closed, the NETCONF server
                will restart its timer until the 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 directs how a NETCONF server
            reconnects to a NETCONF client, after discovering its
            connection to the client has dropped, even if due to a
            reboot.  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. Design Considerations

Editorial: this section is a hold over from before, previously called "Objectives". It was only written two support the "server" (not the "client"). The question is if it's better to add the missing "client" parts, or remove this section altogether.

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF client and servers. This scope includes the following objectives:

4.1. Support all NETCONF transports

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

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

4.2. 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.

4.3. Support authenticating NETCONF clients certificates

When a certificate is used to authenticate a NETCONF 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.

4.4. Support mapping authenticated NETCONF client certificates to usernames

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

4.5. Support both listening for connections and call home

The NETCONF 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. 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.

4.6. For Call Home connections

The following objectives only pertain to call home connections.

4.6.1. Support more than one NETCONF client

A NETCONF server may be managed by more than one NETCONF 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.

4.6.2. Support NETCONF clients having more than one endpoint

A NETCONF client managing a NETCONF 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.

4.6.3. Support a reconnection strategy

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

4.6.4. Support both persistent and periodic connections

NETCONF clients may vary greatly on how frequently they need to interact with a NETCONF 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.

4.6.5. 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.

4.6.6. 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 client requirements, and therefore keep-alive settings should be configurable on a per-client basis.

4.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the NETCONF server to know how often it should connect. This frequency determines the maximum amount of time a NETCONF 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).

5. Security Considerations

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).

6. IANA Considerations

6.1. 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-client
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-netconf-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

6.2. The YANG Module Names Registry

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

```
      name:         ietf-netconf-client
      namespace:    urn:ietf:params:xml:ns:yang:ietf-netconf-client
      prefix:       ncc
      reference:    RFC XXXX

      name:         ietf-netconf-server
      namespace:    urn:ietf:params:xml:ns:yang:ietf-netconf-server
      prefix:       ncs
      reference:    RFC XXXX
```

7. 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.

8. References

8.1. Normative References

[draft-ietf-netconf-ssh-client-server]
Watsen, K., "SSH Client and Server Models", draft-ieft-netconf-ssh-client-server-00 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-ssh-client-server>.

[draft-ietf-netconf-system-keychain]
Watsen, K., "System Keychain Model", draft-ieft-netconf-system-keychain-00 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-system-keychain>.

[draft-ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ieft-netconf-tls-client-server-00 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-tls-client-server>.

[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>.

[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>.

[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>.

8.2. Informative References

[draft-ietf-netconf-call-home] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ieft-netconf-call-home-17 (work in progress), 2015, <https://datatracker.ietf.org/html/draft-ieft-netconf-call-home-17>.

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

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <http://www.rfc-editor.org/info/rfc4252>.

[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>.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <http://www.rfc-editor.org/info/rfc4254>.

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

Appendix A. Change Log

A.1. server-model-09 to 00

- This draft was split out from draft-ietf-netconf-server-model-09.
- Added in previously missing ietf-netconf-client module.
- Added in new features 'listen' and 'call-home' so future transports can be augmented in.

Appendix B. Open Issues

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

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

K. Watsen
Juniper Networks
J. Schoenwaelder
Jacobs University Bremen
July 8, 2016

RESTCONF Client and Server Models
draft-ietf-netconf-restconf-client-server-00

Abstract

This document defines two YANG modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server. Both modules support the TLS transport protocol with both standard RESTCONF and RESTCONF Call Home connections.

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.

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-system-keychain
- draft-ietf-netconf-tls-client-server

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-restconf
- "ZZZZ" --> the assigned RFC value for draft-ietf-netconf-tls-client-server

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

- "2016-07-08" --> 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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines two YANG [RFC6020] modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server [draft-ietf-netconf-restconf]. Both modules support the TLS [RFC5246] transport protocol with both standard RESTCONF and RESTCONF Call Home connections [draft-ietf-netconf-call-home].

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. The RESTCONF Client Model

EDITOR NOTE: Please ignore this section, it is incomplete.

The RESTCONF client model presented in this section supports both clients initiating connections to servers, as well as clients listening for connections from servers calling home.

This model supports both TLS transport protocols using the TLS client groupings defined in [draft-ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keychain model defined in [draft-ietf-netconf-system-keychain].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF client supports.

2.1. Tree Diagram

Note: all lines are folded at column 71 with no '\' character.

```
module: ietf-restconf-client
   +--rw restconf-client
      +--rw initiate {tls-initiate}?
      +--rw listen {tls-listen}?
```

2.2. Example Usage

The following example illustrates configuring a RESTCONF client to initiate connections, as well as listening for call-home connections.

This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

FIXME

2.3. YANG Model

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

<CODE BEGINS> file "ietf-restconf-client@2016-07-08.yang"

```
   // Editor's Note:
   // This module is incomplete at this time.  Below is
   // just a skeleton so there's something in the draft.
   // Please ignore this module for now!

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

     namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-client";
     prefix "rcc";

   /*
     import ietf-inet-types {
       prefix inet;
       reference
         "RFC 6991: Common YANG Data Types";
     }

     //import ietf-netconf-acm {
     //  prefix nacm;
     //  reference
     //    "RFC 6536: Network Configuration Protocol (NETCONF)
     //     Access Control Model";
     //}

     import ietf-x509-cert-to-name {
       prefix x509c2n;
       reference
         "RFC 7407: A YANG Data Model for SNMP Configuration";
     }

     import ietf-tls-client {
       prefix ts;
       revision-date 2016-07-08; // stable grouping definitions
       reference
         "RFC ZZZZ: TLS Client and Server Models";
     }
   */
     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 XXXX; see
       the RFC itself for full legal notices.";

     revision "2016-07-08" {
       description
        "Initial version";
       reference
        "RFC XXXX: RESTCONF Client and Server Models";
     }

     // Features

     feature tls-initiate {
       description
        "The tls-initiate feature indicates that the RESTCONF client
         supports initiating TLS connections to RESTCONF servers.";
       reference
        "RFC YYYY: RESTCONF Protocol";
     }

     feature tls-listen {
       description
        "The tls-listen feature indicates that the RESTCONF client
         supports opening a port to listen for incoming RESTCONF
         server call-home TLS connections.";
       reference
        "RFC AAAA: NETCONF Call Home and RESTCONF Call Home";
```

```
    }

    container restconf-client {
      description
        "Top-level container for RESTCONF client configuration.";
      container initiate {
        if-feature tls-initiate;
        description
          "Configures client intiating underlying TCP connections.";
      }
      container listen {
        if-feature tls-listen;
        description
          "Configures client accepting call-home TCP connections.";
      }
    }

  }
```

<CODE ENDS>

3. The RESTCONF Server Model

The RESTCONF Server model presented in this section supports servers both listening for connections as well as initiating call-home connections.

This model supports the TLS using the TLS server groupings defined in [draft-ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keychain model defined in [draft-ietf-netconf-system-keychain].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF server supports.

3.1. Tree Diagram

Note: all lines are folded at column 71 with no '\' character.

```
 module: ietf-restconf-server
    +--rw restconf-server
       +--rw listen {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/name
       |              +--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 {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/name
            |        +--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
```

3.2. Example Usage

The following example illustrates 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 the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

```
<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>
```

3.3. YANG Model

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

```
<CODE BEGINS> file "ietf-restconf-server@2016-07-08.yang"

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

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

  //import ietf-netconf-acm {
  //  prefix nacm;
  //  reference
  //    "RFC 6536: Network Configuration Protocol (NETCONF)
  //     Access Control Model";
  //}

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-x509-cert-to-name {
    prefix x509c2n;
    reference
```

```
        "RFC 7407: A YANG Data Model for SNMP Configuration";
    }

    import ietf-tls-server {
      prefix ts;
      revision-date 2016-07-08; // stable grouping definitions
      reference
        "RFC ZZZZ: TLS Client and Server Models";
    }

    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 XXXX; see
      the RFC itself for full legal notices.";

    revision "2016-07-08" {
      description
       "Initial version";
      reference
```

```
       "RFC XXXX: RESTCONF Client and Server Models";
    }


    // Features

    feature listen {
      description
       "The 'listen' feature indicates that the RESTCONF server
        supports opening a port to accept RESTCONF client connections
        using at least one transport (e.g., TLS, etc.).";
    }

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

    feature call-home {
      description
       "The 'call-home' feature indicates that the RESTCONF server
        supports initiating RESTCONF call home connections to REETCONF
        clients using at least one transport (e.g., TLS, etc.).";
      reference
       "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
    }

    feature tls-call-home {
      description
       "The 'tls-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 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 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 4336;
                }
              }
              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 maximum 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 client must close the
                  connection when it's ready to release it.  Once the
                  connection has been closed, the server will restart
                  its timer until the 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 to
                    the RESTCONF client before this time if desired
                    (e.g., to deliver a notification).";
                }
              }
            }
          }
        }
        container reconnect-strategy {
          description
           "The reconnection strategy directs how a RESTCONF server
            reconnects to a RESTCONF client after after discovering
            its connection to the client has dropped, 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>
```

4. Security Considerations

5. IANA Considerations

5.1. 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-restconf-client
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.
```

5.2. The YANG Module Names Registry

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

```
name:         ietf-restconf-client
namespace:    urn:ietf:params:xml:ns:yang:ietf-restconf-client
prefix:       ncc
reference:    RFC XXXX

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

6. 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.

7. References

7.1. Normative References

[draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ieft-netconf-restconf-13 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-restconf>.

[draft-ietf-netconf-system-keychain]
Watsen, K., "System Keychain Model", draft-ieft-netconf-system-keychain-00 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-system-keychain>.

[draft-ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ieft-netconf-tls-client-server-00 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-tls-client-server>.

[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>.

[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>.

[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>.

7.2. Informative References

[draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ieft-netconf-call-home-17 (work in progress), 2015, <https://datatracker.ietf.org/html/draft-ieft-netconf-call-home-17>.

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

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

Appendix A. Change Log

A.1. server-model-09 to 00

- This draft was split out from draft-ietf-netconf-server-model-09.
- Added in new features 'listen' and 'call-home' so future transports can be augmented in.

Appendix B. Open Issues

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

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Networks

July 8, 2016

SSH Client and Server Models
draft-ietf-netconf-ssh-client-server-00

Abstract

This document defines two YANG modules, one defines groupings for a generic SSH client and the other defines groupings for a generic SSH server. It is intended that these groupings will be used by applications using the SSH protocol.

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.

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-system-keychain

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-system-keychain

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

- "2016-07-08" --> the publication date of this draft

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

- Appendix A. Change Log

o Appendix B. Open Issues

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines two YANG [RFC6020] modules, one defines groupings for a generic SSH client and the other defines groupings for a generic SSH server (SSH is defined in [RFC4252], [RFC4253], and [RFC4254]). It is intended that these groupings will be used by applications using the SSH protocol. For instance, these groupings could be used to help define the data model for an OpenSSH [OPENSSH] server or a NETCONF over SSH [RFC6242] based server.

The two YANG modules in this document each define two groupings. One grouping defines everything other than what's needed for the TCP [RFC793] protocol layer. The other grouping uses the first grouping while adding TCP layer specifics (e.g., addresses to connect to, ports to listen on, etc.). This separation is done in order to enable applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [draft-ietf-netconf-call-home] could use the first grouping for the SSH parts it provides, while adding data nodes for the reversed TCP layer.

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. The SSH Client Model

The SSH client model presented in this section contains two YANG groupings, one for a client that initiates the underlying TCP connection and another for a client that has had the TCP connection opened for it already (e.g., call home).

Both of these groupings reference data nodes defined by the System Keychain model [draft-ietf-netconf-system-keychain]. For instance, a reference to the keychain model is made to indicate which trusted CA certificate a client should use to authenticate X.509v3 certificate based host keys [RFC6187].

2.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the ietf-ssh-client module.

```
module: ietf-ssh-client
groupings:
  initiating-ssh-client-grouping
     +---- server-auth
     |  +---- trusted-ssh-host-keys?   -> /kc:keychain/trusted-ssh-host
-keys/name
     |  +---- trusted-ca-certs?        -> /kc:keychain/trusted-certific
ates/name {ssh-x509-certs}?
     |  +---- trusted-server-certs?    -> /kc:keychain/trusted-certific
ates/name
     +---- client-auth
        +---- matches* [name]
           +---- name?                     string
           +---- match* [name]
           |  +---- name?                     string
           |  +---- trusted-ssh-host-keys?   -> /kc:keychain/trusted-ss
h-host-keys/name
           |  +---- trusted-ca-certs?        -> /kc:keychain/trusted-ce
rtificates/name
           |  +---- trusted-server-certs?    -> /kc:keychain/trusted-ce
rtificates/name
           +---- user-auth-credentials?   -> /kc:keychain/user-auth-cre
dentials/user-auth-credential/username

  listening-ssh-client-grouping
     +---- address?       inet:ip-address
     +---- port?          inet:port-number
     +---- server-auth
     |  +---- trusted-ssh-host-keys?   -> /kc:keychain/trusted-ssh-host
-keys/name
     |  +---- trusted-ca-certs?        -> /kc:keychain/trusted-certific
ates/name {ssh-x509-certs}?
     |  +---- trusted-server-certs?    -> /kc:keychain/trusted-certific
ates/name
     +---- client-auth
        +---- matches* [name]
           +---- name?                     string
           +---- match* [name]
           |  +---- name?                     string
           |  +---- trusted-ssh-host-keys?   -> /kc:keychain/trusted-ss
h-host-keys/name
           |  +---- trusted-ca-certs?        -> /kc:keychain/trusted-ce
rtificates/name
           |  +---- trusted-server-certs?    -> /kc:keychain/trusted-ce
rtificates/name
           +---- user-auth-credentials?   -> /kc:keychain/user-auth-cre
dentials/user-auth-credential/username
```

2.2. Example Usage

This section shows how it would appear if the initiating-ssh-client-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

FIXME (how to do an example for a module that only has groupings?)

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [draft-ietf-netconf-system-keychain].

```
<CODE BEGINS> file "ietf-ssh-client@2016-07-08.yang"

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

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-client";
  prefix "sshc";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-system-keychain {
    prefix kc;
    reference
      "RFC YYYY: System Keychain Model";
  }

  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>
```

```
      Author:   Kent Watsen
                <mailto:kwatsen@juniper.net>

      Author:   Gary Wu
                <mailto:garywu@cisco.com>";


    description
     "This module defines a reusable grouping for a SSH client that
      can be used as a basis for specific SSH client 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 XXXX; see
      the RFC itself for full legal notices.";

    revision "2016-07-08" {
      description
       "Initial version";
      reference
       "RFC XXXX: SSH Client and Server Models";
    }

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

    grouping initiating-ssh-client-grouping {
      description
        "A reusable grouping for a SSH client that initiates the
         underlying TCP transport connection.";

      container server-auth {
        description
          "Trusted server identities.";
```

```
        leaf trusted-ssh-host-keys {
          type leafref {
            path "/kc:keychain/kc:trusted-ssh-host-keys/kc:name";
          }
          description
            "A reference to a list of SSH host keys used by the
             SSH client to authenticate SSH server host keys.
             A server host key is authenticate if it is an exact
             match to a configured trusted SSH host key.";
        }

        leaf trusted-ca-certs {
          if-feature ssh-x509-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 client to authenticate
             SSH server certificates.";
        }

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

      container client-auth {
        description
          "The credentials used by the client to authenticate to
           the SSH server.";

        list matches {
          key name;
          description
            "A matches expression, which performs like a firewall
             rulebase in that each matches expression is considered
             for a match before moving onto the next matches
             expression.  The first matching expression terminates
             the search, and its 'user-auth-credentials' are used
             to log into the SSH server.";
```

```
          leaf name {
            type string;
            description
              "An arbitrary name for this matches expression.";
          }
          list match {
            key name;
            description
              "A match rule.  The presented SSH server's host key
               is matched against possible trusted SSH host keys
               and certificates.  If a match is found, the specified
               'user-auth-credentials' is used to log into the
               SSH server.";
            leaf name {
              type string;
              description
                "An arbitrary name for this match rule.";
            }
            leaf trusted-ssh-host-keys {
              type leafref {
                path "/kc:keychain/kc:trusted-ssh-host-keys/kc:name";
              }
              description
                "A test to see if the presented SSH host key
                 matches any of the host keys in the specified
                 'trusted-ssh-host-keys' list maintained by the
                 system-keychain module.";
            }
            leaf trusted-ca-certs {
              type leafref {
                path "/kc:keychain/kc:trusted-certificates/kc:name";
              }
              description
                "A test to see if the presented SSH host key matches
                 any of the trusted CA certificates in the specified
                 'trusted-certificates' list maintained by the
                 system-keychain module.";
            }
            leaf trusted-server-certs {
              type leafref {
                path "/kc:keychain/kc:trusted-certificates/kc:name";
              }
              description
                "A test to see if the presented SSH host key matches
                 any of the trusted server certificates in the specified
                 'trusted-certificates' list maintained by the
                 system-keychain module.";
            }
```

```
          }
          leaf user-auth-credentials {
            type leafref {
              path "/kc:keychain/kc:user-auth-credentials/"
                   + "kc:user-auth-credential/kc:username";
            }
            description
              "The specific user authentication credentials to use if
               all of the above 'match' expressions match.";
          }
        }
      }
    }  // end initiating-ssh-client-grouping


    grouping listening-ssh-client-grouping {
      description
        "A reusable grouping for a SSH client that does not
         the underlying TCP transport connection have been
         established using some other mechanism.";
      leaf address {
        type inet:ip-address;
        description
         "The IP address to listen for call-home connections on.";
      }
      leaf port {
        type inet:port-number;
        description
         "The port number to listen for call-home connections.
          When this grouping is used, it is RECOMMENED that
          refine statement is used to either set a default port
          value or to set mandatory true.";
      }
      uses initiating-ssh-client-grouping;
    }


  }


  <CODE ENDS>
```

3. The SSH Server Model

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

Both of these groupings reference data nodes defined by the System Keychain model [draft-ietf-netconf-system-keychain]. For instance, a reference to the keychain model is made to indicate which host key a server should present.

3.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the ietf-ssh-server module.

```
module: ietf-ssh-server
groupings:
  listening-ssh-server-grouping
     +---- address?            inet:ip-address
     +---- port?               inet:port-number
     +---- host-keys
     |  +---- host-key* [name]
     |     +---- name           string
     |     +---- (type)?
     |        +--:(public-key)
     |        |  +---- public-key?    -> /kc:keychain/private-keys/priv
ate-key/name
     |        +--:(certificate)
     |           +---- certificate?   -> /kc:keychain/private-keys/priv
ate-key/certificate-chains/certificate-chain/name {ssh-x509-certs}?
     +---- client-cert-auth {ssh-x509-certs}?
        +---- trusted-ca-certs?       -> /kc:keychain/trusted-certifica
tes/name
        +---- trusted-client-certs?   -> /kc:keychain/trusted-certifica
tes/name

  non-listening-ssh-server-grouping
     +---- host-keys
     |  +---- host-key* [name]
     |     +---- name           string
     |     +---- (type)?
     |        +--:(public-key)
     |        |  +---- public-key?    -> /kc:keychain/private-keys/priv
ate-key/name
     |        +--:(certificate)
     |           +---- certificate?   -> /kc:keychain/private-keys/priv
ate-key/certificate-chains/certificate-chain/name {ssh-x509-certs}?
     +---- client-cert-auth {ssh-x509-certs}?
        +---- trusted-ca-certs?       -> /kc:keychain/trusted-certifica
tes/name
        +---- trusted-client-certs?   -> /kc:keychain/trusted-certifica
tes/name
```

3.2. Example Usage

This section shows how it would appear if the listening-ssh-server-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

```
   <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>
```

3.3. YANG Model

This YANG module has a normative references to [RFC4253], [RFC6991], and [draft-ietf-netconf-system-keychain].

```
  <CODE BEGINS> file "ietf-ssh-server@2016-07-08.yang"

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

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

    import ietf-inet-types {
      prefix inet;
      reference
        "RFC 6991: Common YANG Data Types";
    }

    import ietf-system-keychain {
      prefix kc;
      reference
        "RFC YYYY: System Keychain Model";
    }

    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 XXXX; see
      the RFC itself for full legal notices.";

    revision "2016-07-08" {
      description
       "Initial version";
      reference
       "RFC XXXX: SSH Client and Server 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:name";
              }
              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.  Please note that some addresses have special
          meanings (e.g., '0.0.0.0' and '::').";
      }
      leaf port {
```

```
        type inet:port-number;
        description
         "The local port number on this interface the SSH server
          listens on.  When this grouping is used, it is RECOMMENED
          that refine statement is used to either set a default port
          value or to set mandatory true.";
      }
      uses non-listening-ssh-server-grouping;
    }
  }


  <CODE ENDS>
```

4. Security Considerations

5. IANA Considerations

5.1. 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-ssh-client
   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-ssh-server
   Registrant Contact: The NETCONF WG of the IETF.
   XML: N/A, the requested URI is an XML namespace.
```

5.2. The YANG Module Names Registry

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

```
   name:         ietf-ssh-client
   namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-client
   prefix:       sshc
   reference:    RFC XXXX

   name:         ietf-ssh-server
   namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-server
   prefix:       sshs
   reference:    RFC XXXX
```

6. 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, Juergen Schoenwaelder, Phil Shafer, Sean Turner, and Bert Wijnen.

7. References

7.1. Normative References

[draft-ietf-netconf-system-keychain]
Watsen, K., "System Keychain Model", draft-ieft-netconf-system-keychain-00 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-system-keychain>.

[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>.

[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>.

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

7.2. Informative References

[draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ieft-netconf-call-home-17 (work in progress), 2015, <https://datatracker.ietf.org/html/draft-ieft-netconf-call-home-17>.

[OPENSSH] "OpenSSH", 2016, <http://www.openssh.com>.

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

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <http://www.rfc-editor.org/info/rfc4252>.

[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>.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <http://www.rfc-editor.org/info/rfc4254>.

[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>.

[RFC793] Postel, J., "TRANSMISSION CONTROL PROTOCOL", STD 7, September 1981, <https://www.ietf.org/rfc/rfc793.txt>.

Appendix A. Change Log

A.1. server-model-09 to 00

- This draft was split out from draft-ietf-netconf-server-model-09.
- Added in previously missing ietf-ssh-client module.
- Noted that '0.0.0.0' and '::' might have special meanings.

Appendix B. Open Issues

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

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Networks

July 8, 2016

System Keychain Model
draft-ietf-netconf-system-keychain-00

Abstract

This document defines a YANG data module for a system-level keychain mechanism, that might be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

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.

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:

- "2016-07-08" --> 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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a YANG [RFC6020] data module for a system-level keychain mechanism, which can be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

This module provides a centralized location for security sensitive data, so that the data can be then referenced by other modules. There are two types of data that are maintained by this module:

- Private keys, and any associated public certificates.

- Sets of trusted certificates.

This document extends special consideration for systems that have Trusted Protection Modules (TPMs). These systems are unique in that the TPM must be directed to generate new private keys (it is not possible to load a private key into a TPM) and it is not possible to backup/restore the TPM's private keys as configuration.

It is not required that a system has an operating system level keychain utility to implement this module.

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. The System Keychain Model

The system keychain module 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., a key associated to an IDevID [Std-802.1AR-2009] certificate), 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-case 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.

2.1. Overview

The system keychain module has the following tree diagram. Please see Section 1.2 for information on how to interpret this diagram.

```
module: ietf-system-keychain
   +--rw keychain
      +--rw private-keys
      |  +--rw private-key* [name]
      |  |  +--rw name                                    string
      |  |  +--ro algorithm?                              identityref
      |  |  +--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 algorithm     identityref
      |  |     +---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
      +--rw trusted-ssh-host-keys* [name]
      |  +--rw name                 string
      |  +--rw description?         string
      |  +--rw trusted-host-key* [name]
      |     +--rw name         string
      |     +--rw host-key     binary
      +--rw user-auth-credentials
         +--rw user-auth-credential* [username]
            +--rw username       string
            +--rw auth-method* [priority]
               +--rw priority               uint8
               +--rw (auth-type)?
                  +--:(certificate)
                  |  +--rw certificate*           -> /keychain/private-
keys/private-key/certificate-chains/certificate-chain/name
                  +--:(public-key)
                  |  +--rw public-key*            -> /keychain/private-
keys/private-key/name
                  +--:(ciphertext-password)
                  |  +--rw ciphertext-password?   string
                  +--:(cleartext-password)
                     +--rw cleartext-password?    string
notifications:
   +---n certificate-expiration
      +--ro certificate        instance-identifier
      +--ro expiration-date    yang:date-and-time
```

2.2. 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/load-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>my-rsa-user-key</name>
      <algorithm>rsa</algorithm>
      <public-key>
        cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
        mJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNkYmZ2aXNiZGZpYmhzZG87Zm
        JvO3NkZ25iO29pLmR6Zgo=
      </public-key>
      <certificate-chains>
        <certificate-chain>
          <name>my-rsa-chain</name>
          <certificate>
            ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
            diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
            LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
            KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
            0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
            FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
            GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
            mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
            RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
            rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
            TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
            c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
            SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
          </certificate>
        </certificate-chain>
      </certificate-chains>
    </private-key>

    <private-key>
      <name>my-ec-user-key</name>
      <algorithm>secp256r1</algorithm>
      <public-key>
        mJpdmhzZGZpbHVidjtvc2lkZmhidml1bHNkYmZ2aXNiZGZpYmhzZG87Zm
        cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2RmanZvO3NkZ
```

```
          JvO3NkZ25iO29pLmR6Zgo=
        </public-key>
        <certificate-chains>
          <certificate-chain>
            <name>my-ec-chain</name>
            <certificate>
              0F3SUJBZ0lKQUpRT2t3bGpNK2pjTUEwR0NTcUdTSWIzRFFFQkJRVU
              ZKY1o2WURiR0lPNDB4ajlPb3JtREdsRUNCVTFvVGlrTmpBME1Rc3d
              diR1V4RXpBUkJnTlZCQU1UQ2tOU1RDQkpjM04xWlhJd2daOHdEUVl
              LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
              KS29aSWh2Y04KQVFFQkJRQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
              FNRFF4Q3pBSkJnTlYKQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
              GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3Z1NYTnpkV1Z5TUI0WApE
              mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
              RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIzRFFFQgpCUVVBQTRHQkFMMmx
              rWmFGNWcyaGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFlS3Bx
              TXp4YXJCbFpDSHlLCklVbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
              c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1I1aDZmazcrQzQ1QXg1RWV
              SWM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
            </certificate>
          </certificate-chain>
        </certificate-chains>
      </private-key>

      <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 (CA certs) 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
          WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW
          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>

    <!-- trusted SSH host keys -->
    <trusted-ssh-host-keys>
      <name>explicitly-trusted-ssh-host-keys</name>
      <description>
        Trusted SSH host keys used to authenticate SSH servers.
        These host keys would be analogous to those stored in
        a known_hosts file in OpenSSH.
      </description>
      <trusted-host-key>
        <name>corp-fw1</name>
        <host-key>
          VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER
          NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd
          WpiMjB2WlhoaGJYQnNaUzVqY215aU9L=
        </host-key>
      </trusted-host-key>
    </trusted-ssh-host-keys>

    <!-- user credentials and associated authentication methods -->
    <user-auth-credentials>
      <user-auth-credential>
        <username>admin</username>
        <auth-method>
```

```
            <priority>1</priority>
            <certificate-chain>my-ec-chain</certificate-chain>
            <certificate-chain>my-rsa-chain</certificate-chain>
          </auth-method>
          <auth-method>
            <priority>2</priority>
            <public-key>my-rsa-user-key</public-key>
          </auth-method>
        </user-auth-credential>
        <user-auth-credential>
          <username>tester</username>
          <auth-method>
            <priority>1</priority>
            <cleartext-password>testing123</cleartext-password>
          </auth-method>
        </user-auth-credential>
        <user-auth-credential>
          <username>ldevid</username>
          <auth-method>
            <priority>1</priority>
            <certificate-chain>my-ldevid-chain</certificate-chain>
          </auth-method>
        </user-auth-credential>
      </user-auth-credentials>

   </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>
```

2.3. YANG Module

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

<CODE BEGINS> file "ietf-system-keychain@2016-07-08.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 {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  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 "2016-07-08" {
    description
     "Initial version";
    reference
     "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                Models";
  }


  identity key-algorithm {
    description
      "Base identity from which all key-algorithms are derived.";
  }

  identity rsa {
    base key-algorithm;
    description
      "The RSA algorithm.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity secp192r1 {
    base key-algorithm;
    description
      "The secp192r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp256r1 {
    base key-algorithm;
    description
      "The secp256r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp384r1 {
```

```
    base key-algorithm;
    description
      "The secp384r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  identity secp521r1 {
    base key-algorithm;
    description
      "The secp521r1 algorithm.";
    reference
      "RFC5480:
         Elliptic Curve Cryptography Subject Public Key Information.";
  }

  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 identityref {
            base "key-algorithm";
          }
          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.  The
                 name must be a unique across all private keys, not
                 just within this private key.";
            }
            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 algorithm {
            type identityref {
              base "key-algorithm";
            }
```

```
            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.  These certificates
         can be used by a server to authenticate clients, or by clients
         to authenticate servers.  The certificates may be endpoint
         specific or for certificate authorities (to authenticate many
         clients at once.  Each list of certificates SHOULD be specific
         to a purpose, as the list as a whole may be referenced by other
         modules.  For instance, a NETCONF server model might point to
         a list of certificates to use when authenticating client
         certificates.";
      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.  Note, this
           'certificate' is a list in order to encode any
           associated intermediate certificates.";
        leaf name {
          type string;
          description
            "An arbitrary name for this trusted certificate. Must
             be unique across all lists of trusted certificates
             (not just this list) so that a leafref to it from
             another module can resolve to unique values.";
        }
        leaf certificate {  // rename to 'data'?
          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).";
      }
    }
  }

  list trusted-ssh-host-keys {
    key name;
    description
      "A list of trusted host-keys.  These host-keys can be used
       by clients to authenticate SSH servers.  The host-keys are
       endpoint specific.  Each list of host-keys SHOULD be
       specific to a purpose, as the list as a whole may be
       referenced by other modules.  For instance, a NETCONF
       client model might point to a list of host-keys to use
       when authenticating servers host-keys.";
    leaf name {
      type string;
      description
        "An arbitrary name for this list of trusted SSH host keys.";
    }
    leaf description {
      type string;
      description
        "An arbitrary description for this list of trusted SSH host
         keys.";
    }
    list trusted-host-key {
      key name;
      description
        "A trusted host key.";
      leaf name {
        type string;
        description
          "An arbitrary name for this trusted host-key. Must be
           unique across all lists of trusted host-keys (not just
           this list) so that a leafref to it from another module
           can resolve to unique values.

           Note that, for when the SSH client is able to listen
           for call-home connections as well, there is no reference
           identifier (e.g., hostname, IP address, etc.) that it
           can use to uniquely identify the server with.  The
           call-home draft recommends SSH servers use X.509v3
```

```
            certificates (RFC6187) when calling home.";
        }
        leaf host-key {  // rename to 'data'?
          type binary;
          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).";
        }
      }
    }

/*
Are the auth credentials truly limited to SSH?
Could they be used by an HTTP client to log into an HTTP server?
If truly just for SSH, maybe rename?
*/
    container user-auth-credentials {
      description
        "A list of user authentication credentials that can be used
         by an SSH client to log into an SSH server, using any of
         the supported authentication methods (e.g., password,
         public key, client certificate, etc.).";
       list user-auth-credential {
         key username;
         description
           "The authentication credentials for a specific user.";
         leaf username {
           type string;
           description
             "The username of this user.  This will be the username
              used, for instance, to log into an SSH server.";
         }
         list auth-method {
           key priority;
           description
             "A method of authenticating as this user.";
           leaf priority {
```

```
            type uint8;
            description
              "When multiple authentication methods in this list are
               supported by the server, the one with the lowest priority
               value will be the one that is used.";
          }
          choice auth-type {
            description
              "The authentication type.";
            leaf-list certificate {
              type leafref {
                path "/keychain/private-keys/private-key/"
                     + "certificate-chains/certificate-chain/name";
              }
              ordered-by user;
              description
                "A list of references to certificates that can be used for
                 user authentication.  When multiple certificates in this
                 list supported by the server, the one that comes
                 before the others in the leaf-list will be used.";
            }
            leaf-list public-key {
              type leafref {
                path "/keychain/private-keys/private-key/name";
              }
              ordered-by user;
              description
                "A list of references to public keys that can be used for
                 user authentication.  When multiple public keys in this
                 list supported by the server, the one that comes
                 before the others in the leaf-list will be used.";
            }
            leaf ciphertext-password {
              type string;
              description
                "An ciphertext password.  The method of encipherment and
                 how that method can be determined from this string is
                 implementation-specific.";
            }
            leaf cleartext-password {
              type string;
              description
                "An cleartext password.";
            }
          }
        }
      }
    }
```

```
  }

  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>

3. Design Considerations

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 by default. This was done to match industry trends in IETF best practice (e.g., matching work being done in TLS 1.3). If 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.

4. 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.

5. IANA Considerations

5.1. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registration is requested:

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

5.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the the following registration is requested:

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

6. 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, Juergen Schoenwaelder; Phil Shafer, Sean Turner, and Bert Wijnen.

7. References

7.1. Normative References

[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>.

[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>.

[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>.

7.2. 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://datatracker.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>.

[Std-802.1AR-2009] IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

Appendix A. Change Log

A.1. server-model-09 to 00

- This draft was split out from draft-ietf-netconf-server-model-09.
- Removed key-usage parameter from generate-private-key action.
- Now /private-keys/private-key/certificates/certificate/name must be globally unique (unique across all private keys).
- Added top-level 'trusted-ssh-host-keys' and 'user-auth-credentials' to support SSH client modules.

Appendix B. Open Issues

Please see: https://github.com/netconf-wg/system-keychain/issues.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

K. Watsen
Juniper Networks

July 8, 2016

TLS Client and Server Models
draft-ietf-netconf-tls-client-server-00

Abstract

This document defines two YANG modules, one defines groupings for a generic TLS client and the other defines groupings for a generic TLS server. It is intended that these groupings will be used by applications using the TLS protocol.

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.

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-system-keychain

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

- "XXXX" --> the assigned RFC value for this draft
- "YYYY" --> the assigned RFC value for draft-ietf-netconf-system-keychain

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

- "2016-07-08" --> 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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines two YANG [RFC6020] modules, one defines groupings for a generic TLS client and the other defines groupings for a generic TLS server (TLS is defined in [RFC5246]). It is intended that these groupings will be used by applications using the TLS protocol. For instance, these groupings could be used to help define the data model for an HTTPS [RFC2818] server or a NETCONF over TLS [RFC7589] based server.

The two YANG modules in this document each define two groupings. One grouping defines everything other than what's needed for the TCP [RFC793] protocol layer. The other grouping uses the first grouping while adding TCP layer specifics (e.g., addresses to connect to, ports to listen on, etc.). This separation is done in order to enable applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [draft-ietf-netconf-call-home] could use the first grouping for the TLS parts it provides, while adding data nodes for the reversed TCP layer.

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. The TLS Client Model

EDITOR NOTE: Please ignore this section, it is incomplete.

The TLS client model presented in this section contains two YANG groupings, one for a client that initiates the underlying TCP connection and another for a client that has had the TCP connection opened for it already (e.g., call home).

Both of these groupings reference data nodes defined by the System Keychain model [draft-ietf-netconf-system-keychain]. For instance, a reference to the keychain model is made to indicate which trusted CA certificate a client should use to authenticate the server's certificate.

2.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the ietf-tls-client module.

```
module: ietf-tls-client
groupings:
  initiating-tls-client-grouping
     +---- some-TBD-tcp-client-stuff?   string
     +---- some-TBD-tls-client-stuff?   string

  non-initiating-tls-client-grouping
     +---- some-TBD-tls-client-stuff?   string
```

2.2. Example Usage

This section shows how it would appear if the initiating-tls-client-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

FIXME

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [draft-ietf-netconf-system-keychain].

<CODE BEGINS> file "ietf-tls-client@2016-07-08.yang"

```
// Editor's Note:
// This module is incomplete at this time.  Below is
// just a skeleton so there's something in the draft.
// Please ignore this module for now!

module ietf-tls-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-client";
  prefix "tlsc";
/*
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-system-keychain {
    prefix kc;
    reference
      "RFC YYYY: System Keychain Model";
  }
*/
  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 client that
       can be used as a basis for specific TLS client 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 XXXX; see
       the RFC itself for full legal notices.";

     revision "2016-07-08" {
       description
        "Initial version";
       reference
        "RFC XXXX: TLS Client and Server Models";
     }

     grouping initiating-tls-client-grouping {
       description
         "A reusable grouping for a TLS client that initiates the
          underlying TCP transport connection.";
       leaf some-TBD-tcp-client-stuff {
         type string;
         description "";
       }
       uses non-initiating-tls-client-grouping;
     }

    grouping non-initiating-tls-client-grouping {
       description
         "A reusable grouping for a TLS client that does not initiate
          the underlying TCP transport connection.";
```

```
      leaf some-TBD-tls-client-stuff {
        type string;
        description "";
      }
    }

  }
```

<CODE ENDS>

3. The TLS Server Model

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

Both of these groupings reference data nodes defined by the System Keychain model [draft-ietf-netconf-system-keychain]. For instance, a reference to the keychain model is made to indicate the certificate a server should present.

3.1. Tree Diagram

The following tree diagram presents the data model for the two groupings defined in the ietf-tls-server module.

```
module: ietf-tls-server
groupings:
  listening-tls-server-grouping
     +---- address?        inet:ip-address
     +---- port?           inet:port-number
     +---- certificates
     |  +---- certificate* [name]
     |     +---- name?   -> /kc:keychain/private-keys/private-key/certi
ficate-chains/certificate-chain/name
     +---- client-auth
        +---- trusted-ca-certs?       -> /kc:keychain/trusted-certifica
tes/name
        +---- trusted-client-certs?   -> /kc:keychain/trusted-certifica
tes/name

  non-listening-tls-server-grouping
     +---- certificates
     |  +---- certificate* [name]
     |     +---- name?   -> /kc:keychain/private-keys/private-key/certi
ficate-chains/certificate-chain/name
     +---- client-auth
        +---- trusted-ca-certs?       -> /kc:keychain/trusted-certifica
tes/name
        +---- trusted-client-certs?   -> /kc:keychain/trusted-certifica
tes/name
```

3.2. Example Usage

This section shows how it would appear if the listening-tls-server-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [draft-ietf-netconf-system-keychain].

```
<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>
```

3.3. YANG Model

This YANG module has a normative references to [RFC6991], and [draft-ietf-netconf-system-keychain].

```
<CODE BEGINS> file "ietf-tls-server@2016-07-08.yang"

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

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
  prefix "tlss";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-system-keychain {
    prefix kc;
    reference
      "RFC YYYY: System Keychain Model";
  }

  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 XXXX; see
        the RFC itself for full legal notices.";

      revision "2016-07-08" {
        description
         "Initial version";
        reference
         "RFC XXXX: TLS Client and Server 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:name";
              }
              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. Please note that some addresses have special
          meanings (e.g., '0.0.0.0' and '::').";
      }
      leaf port {
        type inet:port-number;
        description
         "The local port number on this interface the TLS server
          listens on.  When this grouping is used, it is RECOMMENDED
          that refine statement is used to either set a default port
          value or to set mandatory true.";
      }
      uses non-listening-tls-server-grouping;
    }
  }
```

<CODE ENDS>

4. Security Considerations

5. IANA Considerations

5.1. 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-tls-client
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-tls-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

5.2. The YANG Module Names Registry

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

```
name:         ietf-tls-client
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-client
prefix:       tlsc
reference:    RFC XXXX

name:         ietf-tls-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-server
prefix:       tlss
reference:    RFC XXXX
```

6. 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, Juergen Schoenwaelder, Phil Shafer, Sean Turner, and Bert Wijnen.

7. References

7.1. Normative References

[draft-ietf-netconf-system-keychain]
Watsen, K., "System Keychain Model", draft-ieft-netconf-system-keychain-00 (work in progress), 2016, <https://datatracker.ietf.org/html/draft-ieft-netconf-system-keychain>.

[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>.

[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>.

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

[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>.

7.2. Informative References

[draft-ietf-netconf-call-home] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ieft-netconf-call-home-17 (work in progress), 2015, <https://datatracker.ietf.org/html/draft-ieft-netconf-call-home-17>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

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

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

[RFC793] Postel, J., "TRANSMISSION CONTROL PROTOCOL", STD 7, September 1981, <https://www.ietf.org/rfc/rfc793.txt>.

Appendix A. Change Log

A.1. server-model-09 to 00

- This draft was split out from draft-ietf-netconf-server-model-09.
- Noted that '0.0.0.0' and '::' might have special meanings.

Appendix B. Open Issues

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

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 17, 2016

A. Clemm
A. Gonzalez Prieto
E. Voit
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
June 15, 2016

Subscribing to YANG datastore push updates
draft-ietf-netconf-yang-push-03.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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

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

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. Furthermore, while delivery of updates using notifications is a viable option, some applications desire the ability to stream updates using other transports.

Accordingly, there is a need for a service that allows client applications to dynamically subscribe to updates of a YANG datastore and that allows the server to push those updates, possibly using one of several delivery mechanisms. Additionally, support for 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. The solution builds on top of the Netconf Event Model [I-D.gonzalez-netconf-5277bis] which defines a mechanism for the management of event subscriptions. At its core, the solution that is defined here introduces a new set of event streams including datastore push updates that clients can subscribe to, as well as extensions to the event subscription model that allow to manage policies that define what and when updates are trigged. To this end, the document specifies a YANG data model that augments the YANG data model (and RPCs) defined as part of the NETCONF Event Model.

Specifically, the solution features the following capabilities:

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

- The ability for a server to push back on requested subscription parameters. Because not every server may support every requested update policy 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 push update subscription parameters. For example, some servers may have a lower limit to the period with which they can send updates, or they may not support on-change updates for every piece of data.

- 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. In addition, optional subscription parameters allow to specify which transport should be used to stream updates, and to define QoS extensions that allow to address aspects such as how to prioritize between streams of updates.

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 configured subscription, the receiver may be a different system than the one that configured the subscription.

RPC: Remote Procedure Call

SNMP: Simple Network Management Protocol

Configured 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 configured 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 push update subscription service, which supports the dynamic as well as static (via configuration) 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 MAY include information about why the subscription was not accepted, in order to facilitate converging on an acceptable set of subscription parameters. Similarly, configured 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:

- The subscription model for configuration and management of the subscriptions.
- The ability to provide hints for acceptable subscription parameters, in cases where a subscription desired by a client cannot currently be served.
- 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 augments the event subscription model defined in [I-D.gonzalez-netconf-5277bis] and introduces several new parameters, for example parameters that allow subscribers to specify what to include in an update, what triggers an update, and how to deliver updates.

The subscription model assumes the presence of one or more conceptual perpetual datastreams of continuous updates that can be subscribed to. There are several datastreams with predefined semantics, such as the stream of updates of all operational data or the stream of updates of all config data. In addition, it is possible to define custom streams with customizable semantics. The model includes the list of update datastreams that are supported by a system and available for subscription.

The subscription model augments the NETCONF event subscription model with a set of parameters as follows:

- An encoding for push 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.

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

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

    + One parameter specifies the dampening period, i.e. the interval that must pass before a successive update 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 for that object is only sent once the dampening period has passed, containing the value of the data node that is then valid. Note that the dampening period applies to each object, not the set of all objects that are part of the same subscription. This means that on the first change of an object, an update for that object is immediately sent, regardless of whether or not for another object of the same subscription a dampening period is already in effect.

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

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

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

+ EDITOR'S NOTE: Several semantic variations are conceivable. In one variation, an on-change notification is sent immediately, but successive dampening updates are sent at fixed period intervals, grouping all changes of objects for which changes have occurred since the sending of their last update and the current dampening period.

o 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. The following types of filters are introduced: subtree filters, with the same semantics as defined in [RFC 6241],and XPath filters. In addition, as with any subscription, an RFC 5277 filter MAY be used, 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.

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 update frequency.

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.

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

3.3. On-Change Considerations

On-change subscriptions allow 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 in addition to RFC 5277 filters that apply to any event subscription:

- subtree-filter: A subtree filter specifies a subtree that the subscription refers to. When specified, updates will only concern data nodes from this subtree. Syntax and semantics correspond to that specified for [RFC6241] section 6.
- xpath-filter: An XPath filter specifies an XPath expression applied to the data in an update, assuming XML-encoded data.

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

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

3.6. Push Data Stream and Transport Mapping

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

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:

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

- 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 1: Push example

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

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-change-update xmlns=
       "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>89</subscription-id>
     <time-of-update>2015-03-09T19:14:56Z</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 2: Push example for on change

The equivalent update when requesting json encoding:

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-change-update xmlns=
       "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>89</subscription-id>
     <time-of-update>2015-03-09T19:14:56Z</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 3: 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 4: 2nd push example for on change update

3.7. Subscription management

There are two ways in which subscriptions can be managed, as specified in the NETCONF Event Notifications model: 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.

The following sections describe how subscription management is applied to YANG Push subscriptions.

3.7.1. 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 5: 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 6: 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.

3.7.2. 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.

3.8. Other considerations

3.8.1. Authorization

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

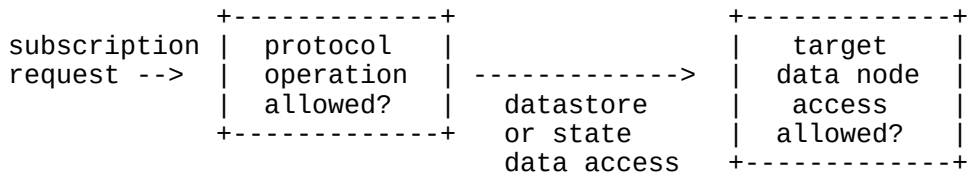


Figure 7: Access control for subscription

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

Another clarification to [RFC6536] is that each of the individual nodes in a pushed update must also go through access control filtering. This includes new nodes added since the last 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.

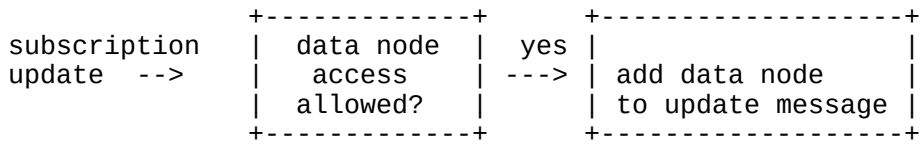


Figure 8: Access control for push updates

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

3.8.2. 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.

3.8.3. 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.

3.8.4. 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.

3.8.5. 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.

3.8.6. 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.

3.8.7. 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
augment /notif-bis:establish-subscription/notif-bis:input:
   +---- subscription-start-time?   yang:date-and-time
   +---- subscription-stop-time?    yang:date-and-time
   +---- (update-trigger)?
   |  +--:(periodic)
   |  |  +---- period                      yang:timeticks
   |  +--:(on-change) {on-change}?
   |     +---- no-synch-on-start?          empty
   |     +---- dampening-period            yang:timeticks
   |     +---- excluded-change*            change-type
   +---- dscp?                        inet:dscp
   |       {notif-bis:configured-subscriptions}?
   +---- subscription-priority?     uint8
   +---- subscription-dependency?   string
augment /notif-bis:establish-subscription/notif-bis:input/
   |       notif-bis:filter-type:
   +--:(update-filter)
      +---- (update-filter)?
         +--:(subtree)
         |  +---- subtree-filter
         +--:(xpath)
            +---- xpath-filter?     yang:xpath1.0
```

```
   augment /notif-bis:establish-subscription/notif-bis:output:
      +---- subscription-start-time?   yang:date-and-time
      +---- subscription-stop-time?    yang:date-and-time
      +---- (update-trigger)?
      |  +--:(periodic)
      |  |  +---- period                     yang:timeticks
      |  +--:(on-change) {on-change}?
      |     +---- no-synch-on-start?         empty
      |     +---- dampening-period           yang:timeticks
      |     +---- excluded-change*           change-type
      +---- dscp?                       inet:dscp
      |       {notif-bis:configured-subscriptions}?
      +---- subscription-priority?     uint8
      +---- subscription-dependency?   string
   augment /notif-bis:establish-subscription/notif-bis:output/
      |     notif-bis:result/notif-bis:no-success/notif-bis:filter-type:
      +--:(update-filter)
         +---- (update-filter)?
            +--:(subtree)
            |  +---- subtree-filter
            +--:(xpath)
               +---- xpath-filter?     yang:xpath1.0
   augment /notif-bis:modify-subscription/notif-bis:input:
      +---- subscription-start-time?   yang:date-and-time
      +---- subscription-stop-time?    yang:date-and-time
      +---- (update-trigger)?
      |  +--:(periodic)
      |  |  +---- period                     yang:timeticks
      |  +--:(on-change) {on-change}?
      |     +---- no-synch-on-start?         empty
      |     +---- dampening-period           yang:timeticks
      |     +---- excluded-change*           change-type
      +---- dscp?                       inet:dscp
      |       {notif-bis:configured-subscriptions}?
      +---- subscription-priority?     uint8
      +---- subscription-dependency?   string
   augment /notif-bis:modify-subscription/notif-bis:input/
      |     notif-bis:filter-type:
      +--:(update-filter)
         +---- (update-filter)?
            +--:(subtree)
            |  +---- subtree-filter
            +--:(xpath)
               +---- xpath-filter?     yang:xpath1.0
   augment /notif-bis:modify-subscription/notif-bis:output:
      +---- subscription-start-time?   yang:date-and-time
      +---- subscription-stop-time?    yang:date-and-time
      +---- (update-trigger)?
```

```
      |  +--:(periodic)
      |  |  +---- period                    yang:timeticks
      |  +--:(on-change) {on-change}?
      |     +---- no-synch-on-start?        empty
      |     +---- dampening-period          yang:timeticks
      |     +---- excluded-change*          change-type
      +---- dscp?                       inet:dscp
      |       {notif-bis:configured-subscriptions}?
      +---- subscription-priority?      uint8
      +---- subscription-dependency?    string
   augment /notif-bis:modify-subscription/notif-bis:output/
      |     notif-bis:result/notif-bis:no-success/notif-bis:filter-type:
      +--:(update-filter)
         +---- (update-filter)?
            +--:(subtree)
            |  +---- subtree-filter
            +--:(xpath)
               +---- xpath-filter?     yang:xpath1.0
   augment /notif-bis:subscription-started:
      +---- subscription-start-time?   yang:date-and-time
      +---- subscription-stop-time?    yang:date-and-time
      +---- (update-trigger)?
      |  +--:(periodic)
      |  |  +---- period                    yang:timeticks
      |  +--:(on-change) {on-change}?
      |     +---- no-synch-on-start?        empty
      |     +---- dampening-period          yang:timeticks
      |     +---- excluded-change*          change-type
      +---- dscp?                       inet:dscp
      |       {notif-bis:configured-subscriptions}?
      +---- subscription-priority?      uint8
      +---- subscription-dependency?    string
   augment /notif-bis:subscription-started/notif-bis:filter-type:
      +--:(update-filter)
         +---- (update-filter)?
            +--:(subtree)
            |  +---- subtree-filter
            +--:(xpath)
               +---- xpath-filter?     yang:xpath1.0
   augment /notif-bis:subscription-modified:
      +---- subscription-start-time?   yang:date-and-time
      +---- subscription-stop-time?    yang:date-and-time
      +---- (update-trigger)?
      |  +--:(periodic)
      |  |  +---- period                    yang:timeticks
      |  +--:(on-change) {on-change}?
      |     +---- no-synch-on-start?        empty
      |     +---- dampening-period          yang:timeticks
```

```
      |     +---- excluded-change*           change-type
      +---- dscp?                       inet:dscp
      |       {notif-bis:configured-subscriptions}?
      +---- subscription-priority?     uint8
      +---- subscription-dependency?   string
   augment /notif-bis:subscription-modified/notif-bis:filter-type:
      +--:(update-filter)
         +---- (update-filter)?
            +--:(subtree)
            |  +---- subtree-filter
            +--:(xpath)
               +---- xpath-filter?     yang:xpath1.0
   augment /notif-bis:filters/notif-bis:filter/notif-bis:filter-type:
      +--:(update-filter)
         +--rw (update-filter)?
            +--:(subtree)
            |  +--rw subtree-filter
            +--:(xpath)
               +--rw xpath-filter?     yang:xpath1.0
   augment /notif-bis:subscription-config/notif-bis:subscription:
      +--rw subscription-start-time?   yang:date-and-time
      +--rw subscription-stop-time?    yang:date-and-time
      +--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 dscp?                       inet:dscp
      |    {notif-bis:configured-subscriptions}?
      +--rw subscription-priority?     uint8
      +--rw subscription-dependency?   string
   augment /notif-bis:subscription-config/notif-bis:subscription/
      |      notif-bis:filter-type:
      +--:(update-filter)
         +--rw (update-filter)?
            +--:(subtree)
            |  +--rw subtree-filter
            +--:(xpath)
               +--rw xpath-filter?     yang:xpath1.0
   augment /notif-bis:subscriptions/notif-bis:subscription:
      +--ro subscription-start-time?   yang:date-and-time
      +--ro subscription-stop-time?    yang:date-and-time
      +--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
      |                          {notif-bis:configured-subscriptions}?
      +--ro subscription-priority?     uint8
      +--ro subscription-dependency?   string
   augment /notif-bis:subscriptions/notif-bis:subscription/
      |      notif-bis:filter-type:
      +--:(update-filter)
         +--ro (update-filter)?
            +--:(subtree)
            |  +--ro subtree-filter
            +--:(xpath)
               +--ro xpath-filter?     yang:xpath1.0
   notifications:
      +---n push-update
      |  +--ro subscription-id            notif-bis:subscription-id
      |  +--ro time-of-update?            yang:date-and-time
      |  +--ro (encoding)?
      |     +--:(encode-xml)
      |     |  +--ro datastore-contents-xml?    datastore-contents-xml
      |     +--:(encode-json) {notif-bis:json}?
      |        +--ro datastore-contents-json?   datastore-contents-json
      +---n push-change-update {on-change}?
         +--ro subscription-id           notif-bis:subscription-id
         +--ro time-of-update?           yang:date-and-time
         +--ro (encoding)?
            +--:(encode-xml)
            |  +--ro datastore-changes-xml?    datastore-changes-xml
            +--:(encode-json) {notif-bis:json}?
               +--ro datastore-changes-yang?   datastore-changes-json
```

Figure 9: Model structure

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

4.2. Update streams

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

4.3. Filters

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

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

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

4.4. Subscription configuration

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

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

- "stream" refers to the stream being subscribed to. The subscription model assumes the presence of perpetual and continuous streams of updates. Various streams are defined: "push-update" covers the entire set of YANG data in the 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.

- o 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 10: 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 11: Establish-subscription positive RPC response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, the lack of read authorization on the requested data node, or the inability of the 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 12: 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 13: 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 14: Establish-subscription error response example 2

4.7.2. Modify-subscription RPC

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

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

Figure 15: Modify subscription request

The 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 16: 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 17: 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@2016-06-15.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-event-notifications {
    prefix notif-bis;
  }
  import ietf-5277-netconf {
    prefix notif;
  }
```

```
  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 2016-06-15 {
    description
      "First revision to incorporate RFC 5277-bis.";
    reference "YANG Datastore Push, draft-ietf-netconf-yang-push-03";
  }

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

 /*
  * IDENTITIES
  */

  /* Additional errors for subscription operations */
  identity error-data-not-authorized {
    base notif-bis:error;
```

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

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

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

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

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

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

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

  /*
```

```
   * TYPE DEFINITIONS
   */

  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 filter-id {
```

```
    type uint32;
    description
      "A type to identify filters which can be associated with a
       subscription.";
  }

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

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

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

  typedef encoding {
    type identityref {
      base notif-bis: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 notif-bis: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.";
  }

  grouping update-filter {
    description
      "This groupings defines filters for push updates for a datastore
```

```
       tree.  The filters define which updates are of interest in a
       push update subscription.
       Mixing and matching of multiple filters does not occur
       at the level of this grouping.
       When a push-update subscription is created, the filter can
       be a regular subscription filter, or one of the additional
       filters that are defined in this grouping.";
    choice update-filter {
      description
        "Define filters regarding which data nodes to include
         in push updates";
      case subtree {
        description
          "Subtree filter.";
        anyxml subtree-filter {
          description
            "Subtree-filter used to specify the data nodes targeted
             for subscription within a subtree, or subtrees, of a
             conceptual YANG datastore.
             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.";
        }
      }
    }
  }

  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 push-subscription-info {
      description
        "This grouping describes information concerning a
         push subscription that is need in addition to information
         already included in notif-bis:subscription-info.";
      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.";
      }
    }

    grouping subscription-qos {
      description
        "This grouping describes Quality of Service information
         concerning a subscription.  This information is passed to lower
         layers for transport priortization and treatment";
      leaf dscp {
```

```
      if-feature "notif-bis:configured-subscriptions";
      type inet:dscp;
      default "0";
      description
        "The push update's IP packet transport priority.
         This is made visible across network hops to receiver.
         The transport priority is shared for all receivers of
         a given subscription.";
    }
    leaf subscription-priority {
      type uint8;
      description
        "Relative priority for a subscription.   Allows an underlying
         transport layer perform informed load balance allocations
         between various subscriptions";
    }
    leaf subscription-dependency {
      type string;
      description
        "Provides the Subscription ID of a parent subscription
         without which this subscription should not exist. In
         other words, there is no reason to stream these objects
         if another subscription is missing.";
    }
  }

  augment "/notif-bis:establish-subscription/notif-bis:input" {
    description
      "Define additional subscription parameters that apply
       specifically to push updates";
    uses push-subscription-info;
    uses update-policy;
    uses subscription-qos;
  }
  augment "/notif-bis:establish-subscription/notif-bis:input/notif-bis:filter-ty
pe" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
  augment "/notif-bis:establish-subscription/notif-bis:output" {
    description
      "Allow to return additional subscription parameters that apply
       specifically to push updates.";
    uses push-subscription-info;
```

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

```
  notification push-update {
    description
      "This notification contains a 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 notif-bis:subscription-id;
      mandatory true;
      description
        "This references the subscription because of which the
         notification is sent.";
    }
    leaf time-of-update {
      type yang:date-and-time;
      description
        "This leaf contains the time of the update.";
    }
    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 "notif-bis: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 notif-bis:subscription-id;
      mandatory true;
      description
        "This references the subscription because of which the
         notification is sent.";
    }
    leaf time-of-update {
      type yang:date-and-time;
      description
        "This leaf contains the time of the update, i.e. the
         time at which the change was observed.";
    }
    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 "notif-bis: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.";
        }
      }
```

```
    }
  }
  augment "/notif-bis:subscription-started" {
    description
      "This augmentation adds push subscription parameters
       to the notification that a subscription has
       started and data updates are beginning to be sent.
       This notification shall only be sent to receivers
       of a subscription; it does not constitute a general-purpose
       notification.";
    uses push-subscription-info;
    uses update-policy;
    uses subscription-qos;
  }
  augment "/notif-bis:subscription-started/notif-bis:filter-type" {
    description
      "This augmentation allows to include additional update filters
       options to be included as part of the notification that a
       subscription has started.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
  augment "/notif-bis:subscription-modified" {
    description
      "This augmentation adds push subscription parameters
       to the notification that a subscription has
       been modified.
       This notification shall only be sent to receivers
       of a subscription; it does not constitute a general-purpose
       notification.";
    uses push-subscription-info;
    uses update-policy;
    uses subscription-qos;
  }
  augment "/notif-bis:subscription-modified/notif-bis:filter-type" {
    description
      "This augmentation allows to include additional update filters
       options to be included as part of the notification that a
       subscription has been modified.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
```

```
  augment "/notif-bis:filters/notif-bis:filter/notif-bis:filter-type" {
    description
      "This container adds additional update filter options
       to the list of configurable filters
       that can be applied to subscriptions.  This facilitates
       the reuse of complex filters once defined.";
    case update-filter {
      uses update-filter;
    }
  }
  augment "/notif-bis:subscription-config/notif-bis:subscription" {
    description
      "Contains the list of subscriptions that are configured,
       as opposed to established via RPC or other means.";
    uses push-subscription-info;
    uses update-policy;
    uses subscription-qos;
  }
  augment "/notif-bis:subscription-config/notif-bis:subscription/notif-bis:filte
r-type" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      uses update-filter;
    }
  }
  augment "/notif-bis:subscriptions/notif-bis:subscription" {
    description
      "Contains the list of currently active subscriptions,
       i.e. subscriptions that are currently in effect,
       used for subscription management and monitoring purposes.
       This includes subscriptions that have been setup via RPC
       primitives, e.g. establish-subscription, delete-subscription,
       and modify-subscription, as well as subscriptions that
       have been established via configuration.";
    uses push-subscription-info;
    uses update-policy;
    uses subscription-qos;
  }
  augment "/notif-bis:subscriptions/notif-bis:subscription/notif-bis:filter-type
" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
}
```

<CODE ENDS>

6. Security Considerations

Subscriptions could be used to attempt to overload 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. Issues that are currently being worked and resolved

7.1. Unresolved issues under discussion

- Which stream types to introduce. Current list includes streams for all operational and for all config data. Consider adding stream for operational data minus counters. Also: assess implications of opstate implications on required data streams.

- In addition to identifying which items go to which streams, identifying and calling out which items (such as counters) should not be "on-change subscribable" may be useful. Consider introducing a Yang extension to define if an object: is-a-counter and/or not-notifiable.

- What QoS parameters should be supported for subscriptions. Note: QoS parameters are applicable to buffering as well as temporarily loss of transport connectivity.

- Implications of ephemeral requirements from I2RS

- Filters: YANG 1.1 allows filters to be defined in multiple places. How do they intersect each other in a deterministic way.

- On-change subscription: consider providing publisher with capability to initiate a refresh of contents rather than send deltas. Current proposal allows for a "synch-on-start" option; such an option might be useful also e.g. on resumption of a subscription that had been suspended.

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

7.2. Agreement in principal

Still need to agree on draft text

- Multiple receivers per configured subscription is ok.

- Proper behavior for on-change, including detecting and indicating changes within a dampening period.

- Still some details to work through, e.g., do we add a counter for the number of object changes during a dampening period?

- Negotiate vs. auto-adjust. Cases where negotiate is needed exists for domain synchronization. Error messages can be used to transport information back as hints. Alternative is dedicated RPC responses. In either case specific contents of these negotiation messages must still be defined.

- Message format for synchronization (i.e. synch-on-start) still needs to be defined.

7.3. Closed Issues

Some editorial updates needed to reflect these

- Periodic interval goes to seconds from timeticks

- Balancing Augment vs. Parallel Model structures (maximize augment)

- o Moving from separate start/stop to Anchor time for Periodic

8. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Yang Geng, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, and Guangying Zheng.

9. References

9.1. 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>.

9.2. 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-04 (work in progress), March 2016.

[I-D.gonzalez-netconf-5277bis]
Gonzalez Prieto, A., Clemm, A., Voit, E., Tripathy, A., Nilsen-Nygaard, E., Chisholm, S., and H. Trevino, "Subscribing to YANG-Defined Event Notifications", draft-clemm-netmod-mount-03 (work in progress), June 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-09 (work in progress), May 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-13, April 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-10 (work in progress), March 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
Internet-Draft
Intended status: Standards Track
Expires: October 8, 2016

K. Watsen
Juniper Networks
M. Abrahamsson
T-Systems

April 6, 2016

Zero Touch Provisioning for NETCONF or RESTCONF based Management
draft-ietf-netconf-zerotouch-08

Abstract

This draft presents a secure technique for establishing a NETCONF or RESTCONF connection between a newly deployed device, configured with just its factory default settings, and its deployment specific network management system (NMS).

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-call-home
- draft-ietf-netconf-restconf
- draft-ieft-netconf-server-model
- draft-pritikin-anima-bootstrapping-keyinfra

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

- "XXXX" --> the assigned 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:

- "2016-04-06" --> the publication date of this draft

The following one Appendix section is to be removed prior to publication:

o Appendix A. Change Log

Status of This Memo

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 to do installations is both cost prohibitive and does not scale.

This document defines bootstrapping strategies enabling devices to securely obtain bootstrapping data with no installer input, beyond physical placement and connecting network and power cables. The ultimate goal of this document is to enable a secure NETCONF [RFC6241] or RESTCONF [draft-ietf-netconf-restconf] connection to the deployment specific network management system (NMS).

1.1. Use Cases

- 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 off of.

- 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 off of. If

no such information is available, or the device is unable to use the information provided, it can then reach out to network just as it would for the remotely administered network use-case.

1.2. Terminology

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

This document uses the following terms:

Artifact: The term "artifact" is used throughout to represent the encoded form of any of Bootstrap Information, Redirect Information, Owner Certificate, and Ownership Voucher. The Bootstrap Server defined in this document is purposed to provide these artifacts, but they can also be provided by any other mechanism (removable storage, DHCP server, etc.), secure or not, so long as the principles for when the bootstrapping data needs to be signed is enforced.

Bootstrapping Data: The term "bootstrapping data" is used throughout this document to refer to the collection of data that a device may obtain from any source of bootstrapping data, including a removable storage device, a DHCP server, a DNS server, a Redirect Server, and/or a Bootstrap Server. This data includes both Redirect Information as well as Bootstrap Information.

Bootstrap Information: The term "bootstrap information" is used herein to refer to bootstrapping data that is used to guide a device to install a specific boot-image and commit a specific configuration. This data is formally defined by the "bootstrap-information" container in the YANG module defined in Section 7.4.

Bootstrap Server: The term "bootstrap server" is used within this document to mean any RESTCONF server implementing the YANG module defined in Section 7.4.

Device: The term "device" is used throughout this document to refer to the network element that needs to be bootstrapped. The device is the RESTCONF client to a Bootstrap Server (see above) and, at the end of bootstrapping process, the device is the NETCONF or RESTCONF server to a deployment-specific NMS. See Section 6 for more information about devices.

Initial Secure Device Identifier (IDevID): The term "IDevID" is defined in [Std-802.1AR-2009] as "the Secure Device Identifier (DevID) installed on the device by the manufacturer". By example, an IDevID certificate, signed by the manufacturer may encode a manufacturer assigned unique identifier (e.g., serial number) and a public key matching a private key held within a TPM chip embedded within the device.

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.

Owner: See Rightful Owner.

Owner Certificate: The term "owner certificate" is used in this document to represent an X.509 certificate, signed by the device's manufacturer or delegate, that binds an owner identity to the owner's private key, which the owner can subsequently use to sign artifacts. The owner certificate is used by devices only when validating owner signatures on signed data. This data is formally defined by the "owner-certificate" container in the YANG module defined in Section 7.4.

Ownership Voucher: The term "ownership voucher" is used in this document to represent manufacturer-specific artifact, signed by the device's manufacturer or delegate, binding an owner identity (same as in the Owner Certificate) to one or more device identities (e.g., serial numbers). The ownership voucher is used by devices only when validating owner signatures on signed data. This data is formally defined by the "ownership-voucher" container in the YANG module defined in Section 7.4.

Redirect Information: The term "redirect information" is used herein to refer to bootstrapping data that redirects a device to connect to another Bootstrap Server. This data is formally defined by the "redirect-information" container in the YANG module defined in Section 7.4.

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, to redirect devices to a deployment-specific bootstrap server.

Rightful Owner: The term "rightful owner" is used herein to refer to the person or organization that purchased a device. Ownership is conveyed by a chain of trust established by a sequence of authenticated secure connections and/or Signed Data, as described in Section 2.3.

Signed Data: The term "signed data" is used throughout to mean either Redirect Information or Bootstrap Information that has been signed by a device's Rightful Owner's private key. These artifacts MUST be signed whenever communicated using an unsecured mechanism. Any time data is signed, it MUST be presented along with an Owner Certificate and Ownership Voucher, which themselves do not need to be signed by the Rightful Owner's private key, as they already are signed by the manufacturer.

Unsigned Data: The term "unsigned data" is used throughout to mean either Redirect Information or Bootstrap Information that has not been signed by a device's Rightful Owner's private key. The option to use unsigned data MUST only be available only when the data is obtained over an authenticated secure connection, such as to a Bootstrap Server.

1.3. 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" (read-write) represents configuration data and "ro" (read-only) represents state data.
- 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. Guiding Principles

This section provides overarching principles guiding the solution presented in this document.

2.1. Trust Anchors

A trust anchor is used in cryptography to represent an entity in which trust is implicit and not derived. In public key infrastructure using X.509 certificates, a root certificate is the trust anchor, from which a chain of trust is derived. The solution presented in this document requires that all the entities involved (e.g., devices, bootstrap servers, NMSs) possess specific trust anchors in order to ensure mutual authentication throughout the zero touch bootstrapping process.

2.2. Conveying Trust

A device in its factory default state possesses a limited set of manufacturer specified trust anchors. In this document, there are two types of trust anchors of interest. The first type of trust anchor is used to authenticate a secure (HTTPS) connection to, for instance, a manufacturer-hosted Internet-based bootstrap server. The second type of trust anchor is used to authenticate manufacturer-signed data, such as the owner certificate and ownership voucher artifacts described in this document.

Using the first type of trust anchor, trust is conveyed by the device first authenticating the secure connection to the bootstrap server and then by the device trusting that the server would only provide data that its rightful owner staged for it to find. Thereby the device can trust any information returned from the server.

Using the second type of trust anchor, trust is conveyed by the device first authenticating the owner certificate and ownership voucher and then, using the public key in the owner certificate, the device can authenticate an owner-signed artifact, such as redirect information. Thereby the device can trust any information held within the artifact.

Notably, the server or artifact may contain redirect information that may include include another trust anchor certificate, for a deployment-specific bootstrap server. Since the device is able to trust the data, using one of its preconfigured trust anchors, it can then use the discovered trust anchor to authenticate a secure connection to the deployment-specific bootstrap server.

2.3. Conveying Ownership

The goal of this document is to enable a device to connect with its rightful owner's NMS. This entails the manufacturer being able to track who owns which devices (out of the scope of this document), as well as an ability to convey that information to devices (in scope).

Matching the two ways to convey trust, this document provides two ways to convey ownership, by using a bootstrap server or by using an ownership voucher.

When a device connects to a bootstrap server configured into its factory default configuration, it implicitly trusts that the bootstrap server would only provide data that its rightful owner staged for it to find. That is, ownership is conveyed by the administrator of the bootstrap server (e.g., a manufacturer) taking the onus of ensuring that only data configured by a device's rightful owner is made available to the device. With this approach, the assignment of a device to an owner is ephemeral, as the administrator can reassign a device to another owner at any time.

When a device is presented signed artifacts, it authenticates that its rightful owner provided the artifact by verifying the signature over the artifact using additional artifacts, the owner certificate and ownership voucher. With this approach, ownership is conveyed by the the manufacturer (or delegate) taking the onus of ensuring that the ownership vouchers it issues are accurate. With this approach, the assignment of a device to an owner may be permanent, as the ability to reassign a device to another owner entails revoking the prior assignment, which requires the device having an accurate and securely set clock, which may not be possible for all devices (see Section 8 for information about this).

3. Information Types

This document presumes there exists two types of zero touch information: redirect information and bootstrap information.

Both information types MAY be signed or unsigned, though in some contexts, as described below, the bootstrap information type MUST be signed, as there is not otherwise possible for a device to process it, even in a degraded manner.

Both information types MAY be encoded using various technologies. This document only tries to support the encodings supported by RESTCONF, namely XML and JSON, while leaving extensibility mechanisms in place to support future extensions.

3.1. Redirect Information

Redirect information provides a list of bootstrap servers, where each list entry includes the bootstrap server's hostname or IP address, an optional port, and an optional trust anchor certificate. The redirect information type is formally defined by the "redirect-information" grouping defined in Section 7.4.

As its name suggests, redirect information guides the device to attempt to connect to the specified bootstrap servers, until finding one that it can bootstrap itself off of. Redirect information is primarily distinguished from standard HTTP redirect by its optional inclusion of trust anchors, in which case it may be referred to as a "secure redirect".

Redirect information may be trusted or untrusted. That is, when the redirect information is obtained via a secure connection to a trusted bootstrap server or a signed artifact, it is trusted. In all other cases (e.g., an unsigned artifiact obtain via DHCP, DNS, or removable storage), the redirect information is untrusted.

Trusted redirect information is useful for enabling a device to establish a secure connection to a bootstrap server. Untrusted redirect information is useful for directing a device to a bootstrap server where signed data has been staged for it to obtain.

When the redirect information is trusted and conveys trust anchors, and the device is able establish a secured connection to the specified bootstrap server, using X.509 certificate path validation ([RFC6125], Section 6) to the trust anchor provided in the redirect information, then the device MUST trust the bootstrap server.

When the redirect information is untrusted, the device MUST discard any presented trust anchors and the device MUST connect to the bootstrap server by blindly accepting the bootstrap server's TLS certificate. In this case, the device MUST NOT trust the bootstrap server.

Implications of a device trusting or not trusting a bootstrap server are discussed in Section 4.4.

3.2. Bootstrap Information

Bootstrap information provides all the data necessary for the device to bootstrap itself, in order to be considered ready to be managed. This data includes criteria about the boot image the device MUST be running, an initial configuration the device MUST commit, and an

optional script that, if specified, the device MUST successfully execute. Descriptions for these follow:

- The boot image criteria is used to ensure the device is running a version of software that will be able to understand the configuration and script, if any. The criteria is flexible in that it allows for both an absolute specification of the boot image a device MUST be running, or just a list of YANG modules that the device MUST be able to understand.

- The configuration can configure any aspect of the device but, in order to fulfill the goal of the zero touch bootstrapping process, to establish a NETCONF or RESTCONF connection to the device's deployment specific NMS, the configuration MUST minimally configure an administrator account (e.g., username, SSH public key) that the NMS can use to log into the device with, and configure the device to either listen for inbound NETCONF/RESTCONF connections, or for the device to initiate an outbound NETCONF/RESTCONF call home connection [draft-ietf-netconf-call-home]. The bootstrap information examples provided in Section 7.2.3, Section 7.2.4, and Section 7.3.2 all illustrate a minimal initial configuration.

- The script, if any, is used to perform non-configuration related activities deemed necessary. The script format is manufacturer specific. Requirements for scripts, such as exit status codes, are defined in the "script" node's description statement provided in the YANG module defined in Section 7.4.

It is always permitted for bootstrap information to be signed, even if it was obtained in a secure fashion. If the device is accessing the bootstrap server in an unsecured manner (e.g., from a removable storage device or from an untrusted server), then the bootstrap information MUST be signed.

Devices MUST process bootstrap information as is specified in Section 6.6.

The bootstrap information type is formally defined by the "bootstrap-information" grouping defined in Section 7.4.

4. Sources for Bootstrapping Data

Following are the sources of bootstrapping data that are referenced by the workflows presented in Section 5.3. Other sources of bootstrapping data may be defined in future documents, so long as the principles for when the bootstrapping data needs to be signed are enforced.

Each of the descriptions below show how the bootstrapping data needs to be handled in a manner consistent with the guiding principles in Section 2.

For devices supporting more than one source for bootstrapping data, no particular sequencing order has to be observed, as each source is equally secure, in that the chain of trust always goes back to the same root of trust, the manufacturer. That said, from a privacy perspective, it is RECOMMENDED that a device try to leverage local sources before remote source. For this reason, all the examples used in this document assume a removable storage device is accessed before a DHCP server, which itself is accessed before an Internet-based bootstrap server.

4.1. Removable Storage

A device MAY attempt to acquire bootstrapping data from a directly attached removable storage device. The bootstrapping data MAY be either redirect information or bootstrap information.

If redirect information is provided, it SHOULD be signed, as removable storage devices are not trustworthy. Section 3.1 defines how a device processes signed and unsigned redirect information.

If bootstrap information is provided, it MUST be signed, as removable storage devices are not trustworthy and there is no option to process the data in a degraded manner, unlike as with redirect information.

For the case when the signed bootstrap information is provided, it is notable that even the raw boot image file itself can be on the removable storage device, by letting the URL reference a local file (e.g., file:///path/to/file), making use of the removable storage device a fully self-standing bootstrapping solution.

Note: details such as the format of filesystem and the naming of the files are left to the device’s manufacturer to define.

4.2. DNS Server

A device MAY attempt to acquire bootstrapping data from a DNS server using DNS-based service discovery (DNS-SD) [RFC6763]. Due to DNS packet size limitations the bootstrapping data provided using DNS-SD can only be redirect information (not bootstrap information).

The redirect information provided via DNS-SD SHOULD be signed (i.e., using the owner’s private key), as this document does not define a solution to secure the DNS records using DNSSEC [RFC6698], and

therefore the DNS records are not trustworthy. Section 3.1 defines how a device processes signed and unsigned redirect information.

To use this approach, the device MAY perform DNS-SD via multicast DNS [RFC6762] searching for the service "_zerotouch._tcp.local.". Alternatively the device MAY perform DNS-SD via normal DNS operation, using the domain returned to it from the DHCP server, searching for the service "_zerotouch._tcp.example.com".

The mapping of redirect information onto DNS SRV [RFC2782] and DNS TXT [RFC1035] records is as follows:

- The bootstrap server's hostname or IP address is returned by the "Target" component of the DNS SRV record.

- The bootstrap server's port is returned by the "Port" component of the DNS SRV record.

- The bootstrap server's trust anchor is returned using the key "anchor" in the DNS TXT record with the binary value being the 'gzip' [RFC1951] compression over the redirect-information's "trust-anchor" value. To save additional space, it is RECOMMENDED that the trust anchor certificate uses an elliptical curve algorithm, rather than the larger, more common RSA algorithm.

- The signature over the preceding three values is returned using the key "sig" in the DNS TXT record with the binary value being the 'gzip' compression over the redirect-information's "signature" value.

- The owner certificate is returned using the key "cert" in the DNS TXT record with the binary value being the 'gzip' compression over the redirect-information's "owner-certificate/certificate" value. There isn't enough space to support returning CRLs. To save additional space, it is RECOMMENDED that the owner certificate uses an elliptical curve algorithm, rather than the seemingly ubiquitous RSA algorithm.

- The ownership voucher is returned using the key "voucher" in the DNS TXT record binary value being the 'gzip' compression over the redirect-information's "ownership-voucher/voucher" value. There isn't enough space to support returning CRLs.

The applicability of this approach across vendors is limited due to the ownership voucher being a manufacturer-specific format. This limitation only impacts signed data, when the ownership voucher is used; there is no such limitation when unsigned data is communicated.

4.3. DHCP Server

A device MAY attempt to acquire bootstrapping data from a DHCP server (e.g., using one of the DHCP options defined in Section 9.1). The bootstrapping data MAY be either redirect information or bootstrap information.

If redirect information is provided, it SHOULD be signed, as the DHCP protocol is not a secure protocol. However, if the redirect information is not signed, then the device MUST NOT trust any included trust anchor certificates, which means that the device would have to establish an unsecured connection to the specified bootstrap servers. See Section 3.1 for more about this case.

If bootstrap information is provided, it MUST be signed, as the DHCP protocol is not a secure protocol and there is no option to process the data in a degraded manner, unlike as with redirect information.

For the case when the signed bootstrap information is provided, it is notable that the URL would have to point to another file server (e.g., http://, ftp://, etc.), as DHCP servers do not themselves distribute files.

It is expected that DHCP servers will provide redirect information more often than bootstrap information, since redirect information is more generic, potentially applicable to a large number of devices, with the number limited only by the number of devices listed by the associated ownership voucher. Still, because the ownership voucher is a manufacturer specific format, it is advisable for devices to send the Vendor Class Identifier (option 60) field in their DHCP lease requests, so that the DHCP server doesn’t accidentally hand it another manufacturer’s voucher format.

If it is desired for the DHCP server to return bootstrap information, care should be taken to ensure that bootstrap information is applicable to all the devices that might connect to the DHCP server. The device SHOULD again pass the Vendor Class Identifier (option 60) field in its DHCP lease request. However, if it is desired to return device-specific bootstrap information, then the device SHOULD also send the Client Identifier (option 61) field in its DHCP lease request so that the DHCP server can select the specific bootstrap information that has been staged for that one device.

4.4. Bootstrap Server

A device MAY attempt to acquire bootstrapping data from a trusted Internet-based bootstrap server, a server implementing the RESTCONF API defined by the YANG module provided in Section 7.4. The

bootstrapping data provided by the server MAY be either redirect information or bootstrap information.

Notably, when using the "notification" action defined in Section 7.4, a bootstrap server is not only a source for bootstrapping data, but can also be the consumer of notification messages from devices. These notification messages both enable visibility into the bootstrapping process (e.g., reporting warnings and errors) and well as provide potentially useful completion status information (e.g., the device's SSH host-keys).

If the device is able to trust the bootstrap server, by verifying its TLS server certificate using a preconfigured or learned trust anchor, then the data the device obtains from the bootstrap server MAY be unsigned. Notably, this is the only mechanism defined in this document whereby unsigned bootstrap information (not redirect information) can be used. When the device is able to trust the bootstrap server, it MUST send its IDevID certificate in the form of a client certificate, and MUST send notifications to the bootstrap server, using the "notification" action defined in Section 7.4.

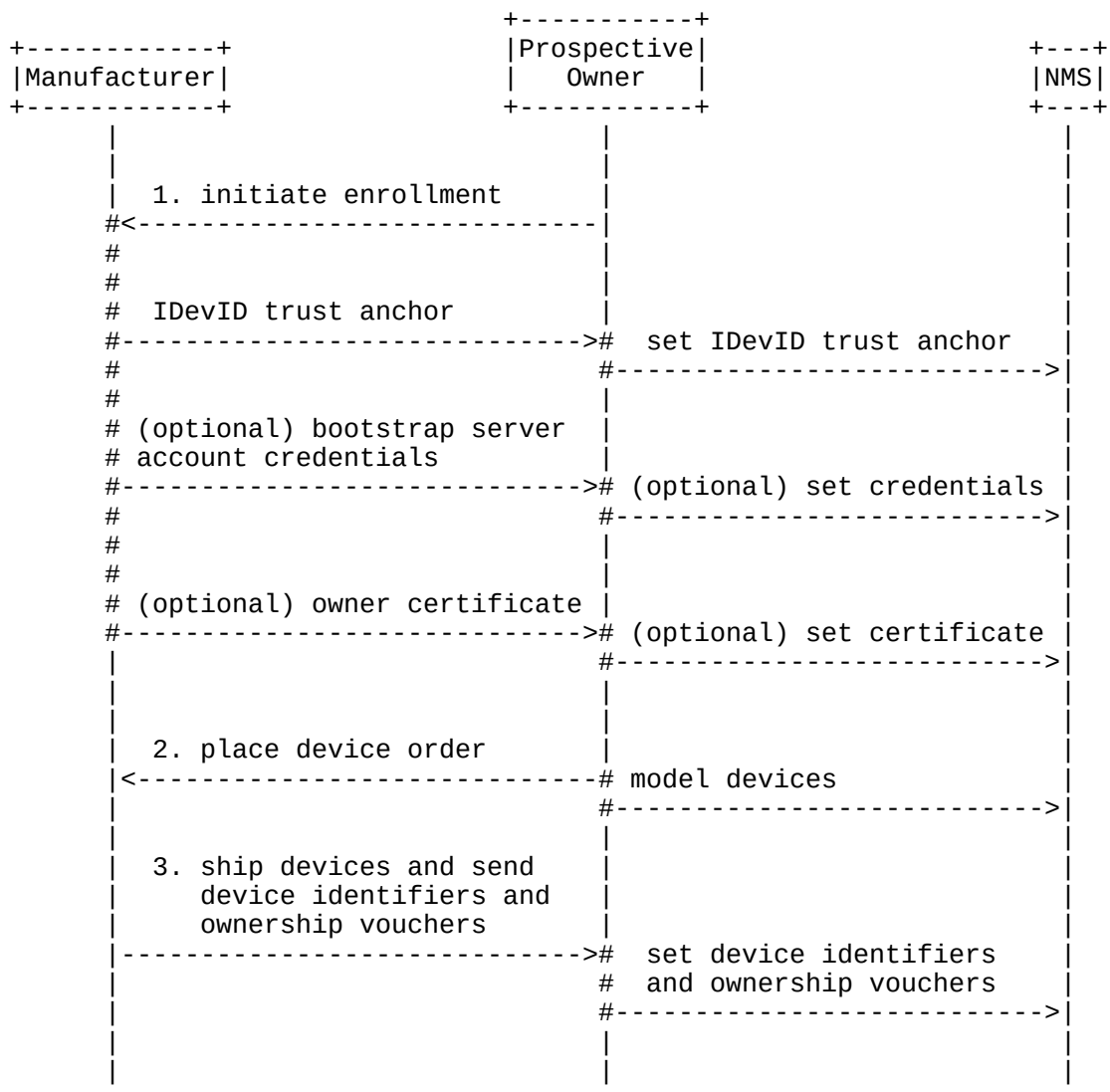
If the device is unable to trust the bootstrap server, then the data the device obtains from the bootstrap server MUST be signed. When the device connects to an untrusted bootstrap server, the device MUST NOT send its IDevID certificate in the form of a client certificate, and MUST NOT send any notifications to the bootstrap server, using the "notification" action defined in Section 7.4.

5. Workflow Overview

The zero touch solution presented in this document is conceptualized to be composed of the workflows described in this section. Implementations MAY vary in details. Each diagram is followed by a detailed description of the steps presented in the diagram, with further explanation on how implementations may vary.

5.1. Onboarding and Ordering Devices

The following diagram illustrates key interactions that occur from when a prospective owner enrolls in a manufacturer's zero touch program to when the manufacturer ships devices for an order placed by the prospective owner.



The interactions in the above diagram are described below.

1. A prospective owner of a manufacturer’s devices, or an existing owner that wishes to start using zero touch for future device orders, would initiate an enrollment process with the manufacturer, or the manufacturer’s delegate.

2.

Regardless how the prospective owner intends to bootstrap their devices, they will always obtain from the manufacturer or delegate the trust anchor certificate needed to authenticate device IDevID certificates. This certificate will need to be installed on the prospective owner's NMS so that the NMS can subsequently authenticate the device's IDevID certificates.

If the manufacturer hosts an Internet based bootstrap 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.

If the manufacturer's devices are able to acquire bootstrapping data from sources other than a manufacturer-hosted Internet-based bootstrap server (e.g., removable storage, DHCP server, etc.), then the manufacturer would additionally provide an owner certificate to the prospective owner. How the owner certificate is used to enable devices to validate signed bootstrapping data is described in Section 6.4. Not depicted, the owner certificate is generated by the prospective owner previously sending a certificate signing request to the manufacturer for signing, thus resulting in the owner certificate. Assuming the prospective owner's NMS is able to prepare and sign the bootstrapping data, the owner certificate would be installed on the NMS at this time.

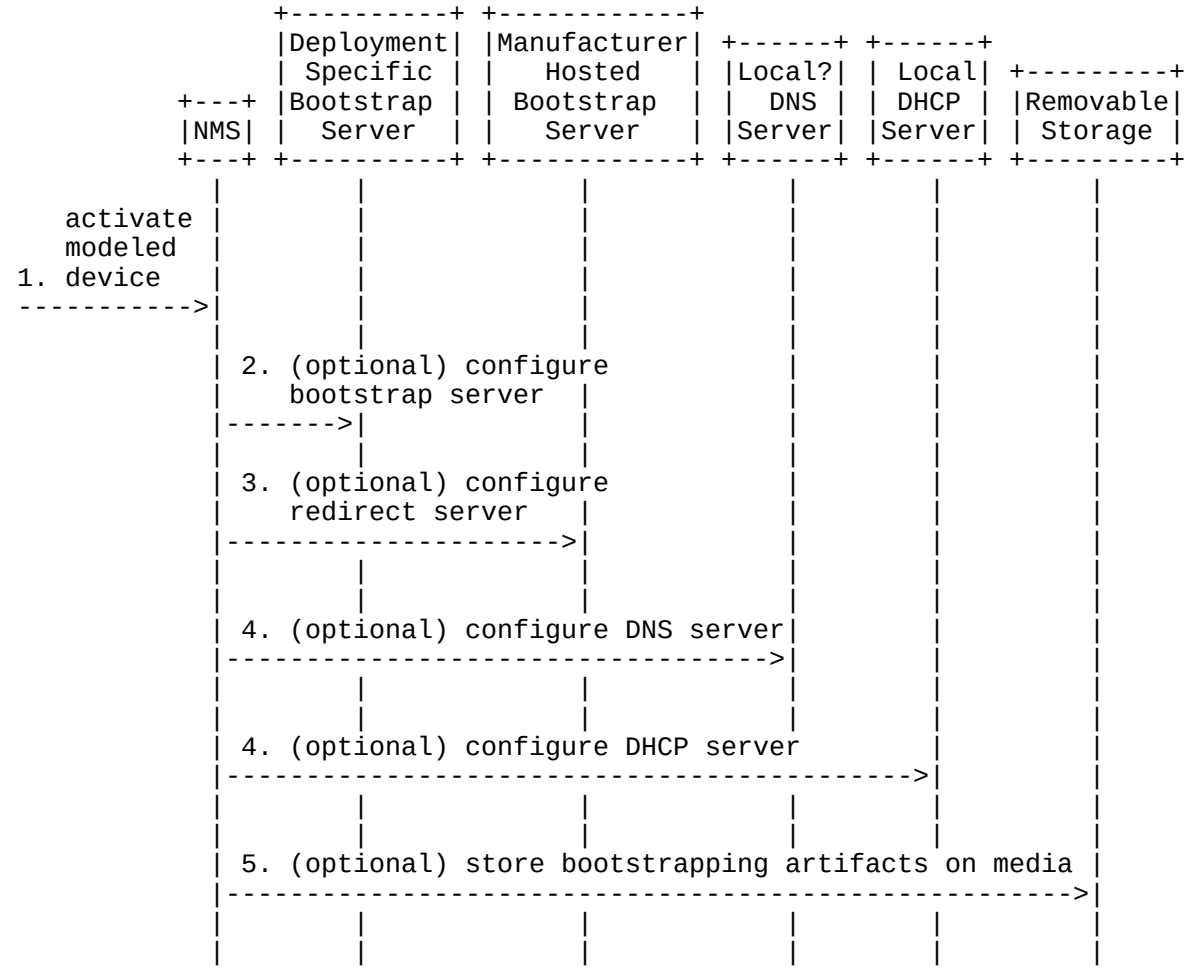
3. Some time later, the prospective owner places an order with the manufacturer, perhaps with a special flag checked for zero touch handling. At this time, or perhaps before placing the order, the owner may model the devices in their NMS. That is, create virtual objects for the devices with no real-world 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 ships the devices for the order, the manufacturer notifies the owner of the devices' unique identifiers and shipping destinations, which the owner can use to stage the network for when the devices powers on. Additionally, the manufacturer may send an ownership voucher, assigning ownership of those devices to the rightful owner. The owner sets this information on their NMS, perhaps binding specific device

identifiers and ownership vouchers (if supported) to specific modeled devices.

5.2. Owner Stages the Network for Bootstrap

The following diagram illustrates how an owner stages the network for bootstrapping devices.



The interactions in the above diagram are described below.

1. Having previously modeled the devices, including setting their fully operational configurations, associating device identifiers and ownership vouchers (if supported), the owner "activates" one or more modeled devices. That is, tell the NMS to perform the

steps necessary to prepare for when the real-world devices are powered 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 information for the specific devices. Whenever a deployment specific bootstrap server is used, the NMS MUST also configure some other source of bootstrapping data (i.e. an Internet based redirect server, a local DHCP server, a removable storage device, etc.) with redirect information, so that the device can discover where the deployment specific server is located and how to establish a connection to it. 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 or delegate hosted bootstrap server, it MUST be configured to provide the bootstrapping information for the specific devices. The configuration MUST be either redirect or bootstrap information. That is, either the manufacturer hosted bootstrap server will redirect the device to another bootstrap server, or provide the device with its bootstrapping information itself. The types of bootstrapping information the manufacturer hosted bootstrap server supports MAY vary by implementation; some implementations may only support redirect information, or only support bootstrap information, or support both redirect and bootstrap 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 information, a DNS server needs to be configured. If multicast DNS-SD is desired, then the server MUST reside on the local network, otherwise the MAY reside on a remote network. Please see Section 4.2 for more information about how to configure DNS servers. Configuring the DHCP server MAY occur via a programmatic API not defined by this document.

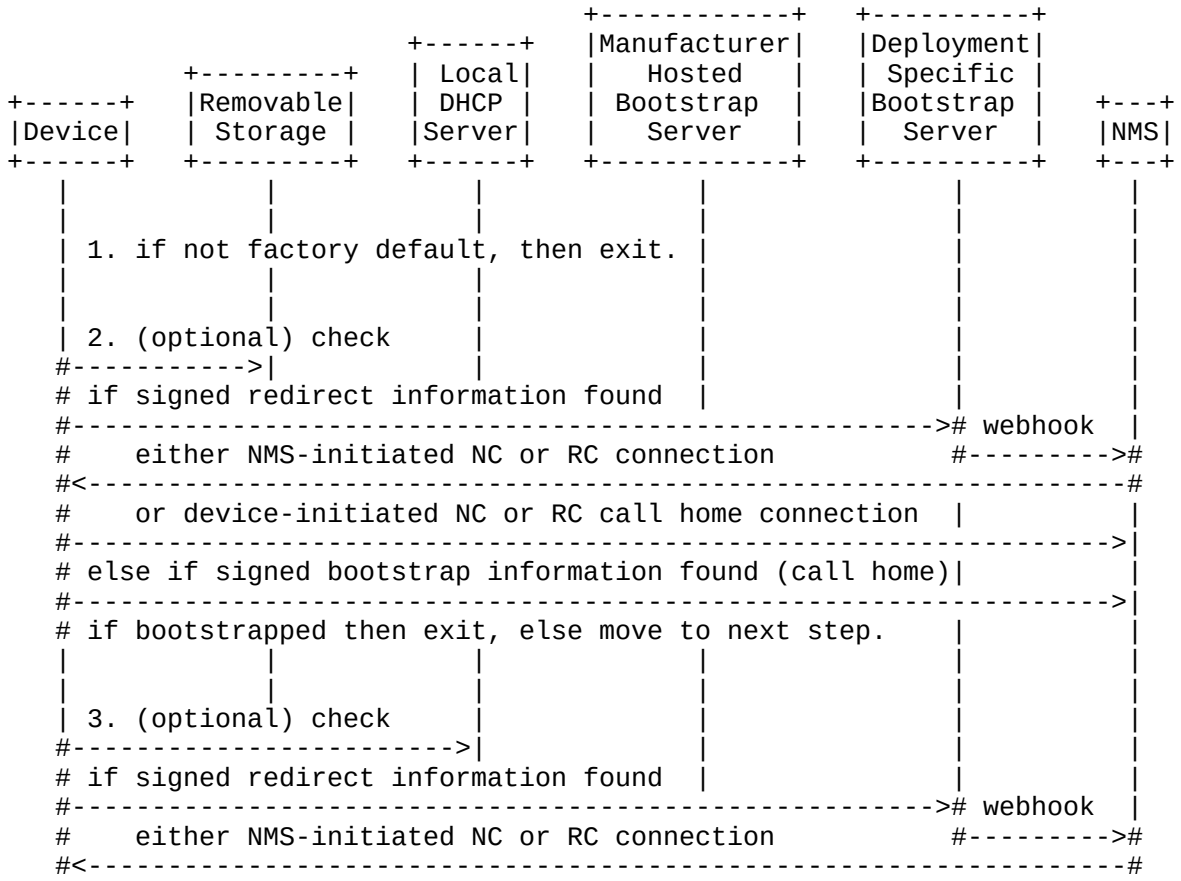
5. If it is desired to use a DHCP server to supply bootstrapping data, the DHCP server MUST be accessible via the network the device is located, either direct or via a DHCP relay. Please see Section 4.3 for more information about how to configure DHCP servers. Configuring the DHCP server MAY occur via a programmatic API not defined by this document.

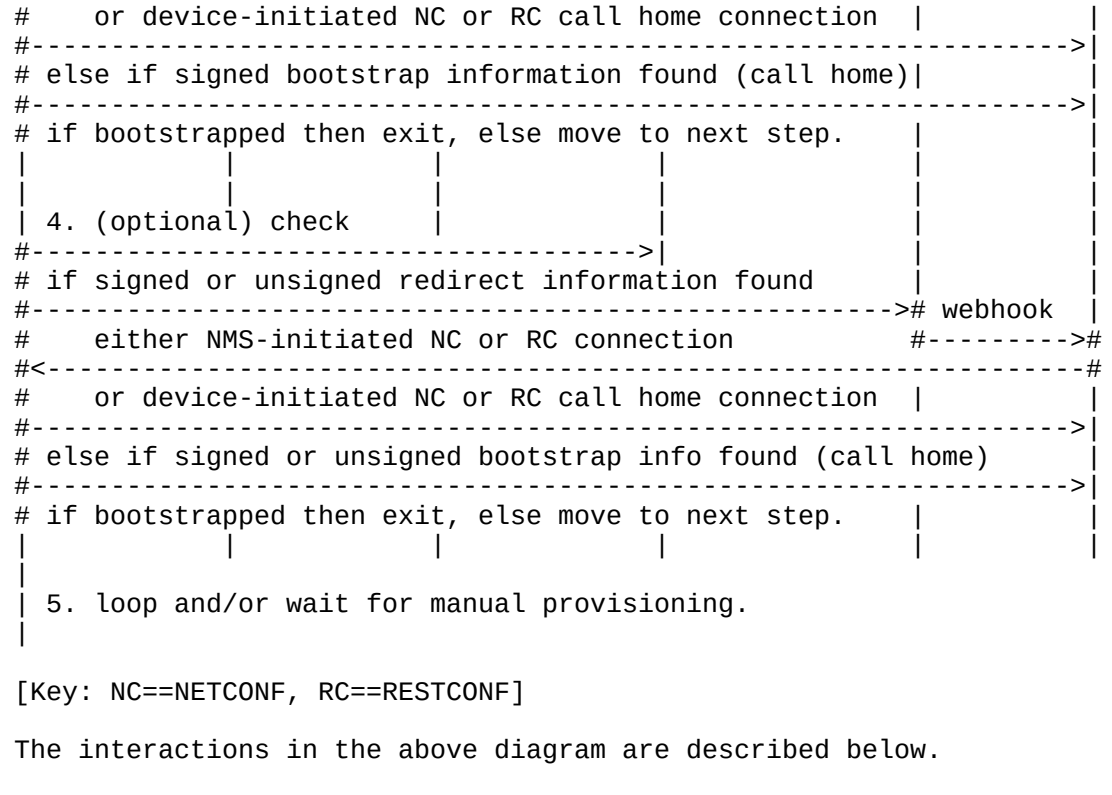
6. If it is desired to use a removable storage device (e.g., USB flash drive) to supply bootstrapping information, the information would need to be placed onto it. Please see Section 4.1 for more information about how to configure a removable storage device.

5.3. Device Powers On

The following diagram illustrates how a device might behave when powered on. Note that this is merely exemplary, subject to which bootstrapping strategies the device supports, which may be more or less than depicted below.

This diagram sequences the sources of bootstrapping information (see Section 4) based on locality, or how "close" the data is to the device, which is RECOMMENDED. Whether this sequence makes sense for a specific type of device needs to be determined by the manufacturer.





[Key: NC==NETCONF, RC==RESTCONF]

The interactions in the above diagram are described below.

1. Upon power being applied, the device's bootstrapping logic first checks to see if it is running in its factory default state. If it has a modified state, then the bootstrapping logic would exit and none to the following interactions would occur.

2. If the device is able to load bootstrapping data from a removable storage device (e.g., USB flash drive), it is RECOMMENDED that it try to do so first. Assuming a removable storage device is attached to the device, the device would check for bootstrapping data and, if found, validate that it has been signed using the procedure described in Section 6.4. The bootstrapping data MAY either be redirect information or bootstrap information. How the device processes each is follows:

   * In the case that redirect information is found (e.g., the example depicted in Section 7.3.1), the device would use the redirect information to establish a secure connection to a deployment-specific bootstrap server. In theory this bootstrap server could return a response that redirected the device to yet another bootstrap server (e.g., the example depicted in Section 7.2.1), but in this example it is depicted

that it returns bootstrap information (e.g., the example depicted in Section 7.2.3). Using this bootstrap information, the device would set its boot image and its initial configuration. If the bootstrap server supports notifying external systems (e.g., via a webhook) when a device has notified the bootstrap server that it is ready to be managed (e.g., the example depicted in Section 7.2.5), it might do so at this time, which could prompt the NMS to initiate a NETCONF or RESTCONF connection to the device at this time. Alternatively, the initial configuration the device installs could configure the device to initiate a NETCONF or RESTCONF call home [draft-ietf-netconf-call-home] connection to the deployment-specific NMS. All of these sub-steps are depicted in the diagram above.

* In the case that bootstrap information is found (e.g., the example depicted in Section 7.2.2), the device would use the bootstrap information to install a boot image, which itself could be located on the same removable storage device, and set its initial configuration. In this case, since there is no easy way to notify the NMS that the device is ready to be managed (e.g., via a webhook), it is RECOMMENDED that the initial configuration directs the device to proactively initiate a NETCONF or RESTCONF call home [draft-ietf-netconf-call-home] connection to the deployment-specific NMS.

If the device is unable to bootstrap using any of the information on the removable storage device, it would proceed to the next source of bootstrapping information, if any.

3. If the device is able to load bootstrapping data from a DHCP server, when obtaining a DHCP assignment, it may receive a response that includes a Zero Touch Information DHCP option (Section 9.1). Details regarding how to process bootstrapping data received from a DHCP server are discussed in Section 4.3.

4. The remainder of the device’s logic is the same as described above for when using a removable storage device. If the device is unable to bootstrap using information provided by a DHCP server, it would proceed to the next source of bootstrapping information, if any.

5. If the device is able to load bootstrapping data from a trusted Internet-based bootstrap server, as preconfigured in its factory default settings (Section 6.1), it is RECOMMENDED that the device attempts to establish a secure TLS connection to the bootstrap server, authenticating its TLS server certificate using the trust

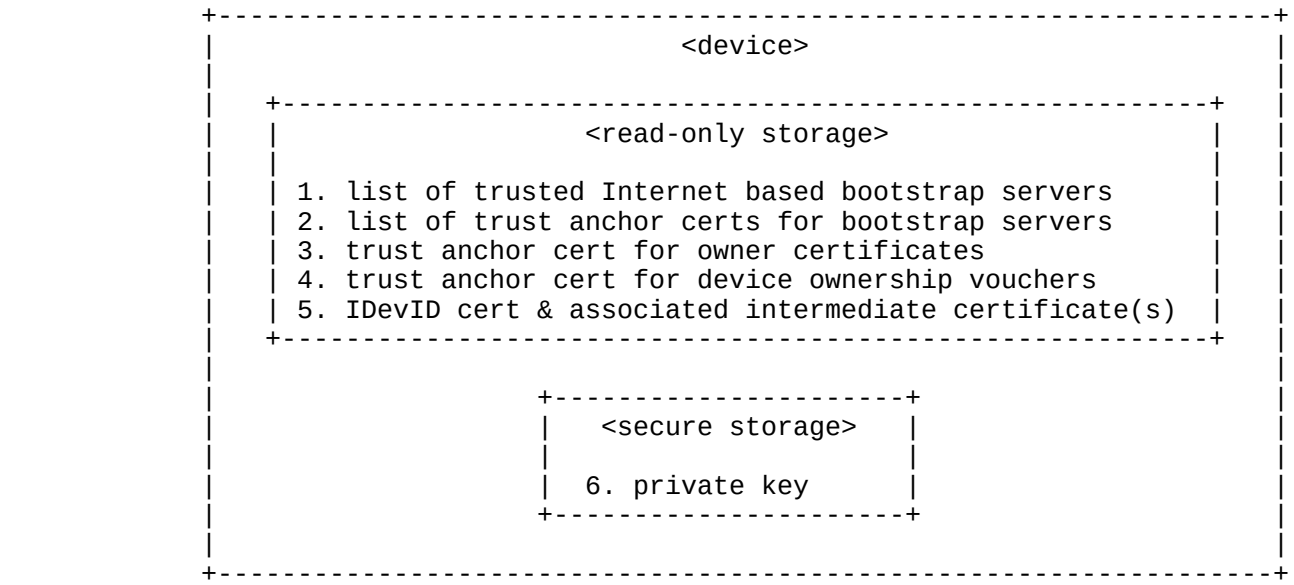
anchors set by its factory default state (Section 6.1), and download any data that has been staged for it there, which MAY not be signed, since the server's certificate could be trusted. In either case, the remainder of the device's logic is the same as described above for when using a removable storage device. If the device is unable to bootstrap using information provided by a DHCP server, it would proceed to the next source of bootstrapping information, if any.

6. If no more sources of bootstrapping information are available, the device MAY retry again all sources of bootstrapping data and/or MAY provide manageability interfaces for manual configuration (e.g., CLI, HTTP, NETCONF, etc.). If manual configuration is allowed, and such configuration is provided, the device MUST immediately cease trying to obtain bootstrapping data, as it would then no longer be in its factory default state.

6. Device Details

Devices supporting Zero Touch MUST have the preconfigured factory default state and bootstrapping logic described in the following sections.

6.1. Factory Default State



Each numbered item below corresponds to a numbered item in the diagram above.

1. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 4) MUST be manufactured with a list of trusted bootstrap servers. Each bootstrap server MAY be identified by just its hostname or IP address, and an optional port. Note that it is not necessary to configure URLs, as the RESTCONF protocol defines how the bootstrap server API specified in Section 7.4 maps into URLs.

2. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 4) SHOULD be manufactured with a list of trust anchor certificates that can be for X.509 certificate path validation [RFC6125], Section 6) on the bootstrap server's TLS server certificate.

3. Devices that support loading owner signed data (see Section 1.2) MUST be manufactured with the trust anchor certificate for the owner certificates that the manufacturer provides to prospective owners when they enroll in the manufacturer's Zero Touch program (see Section 5.1).

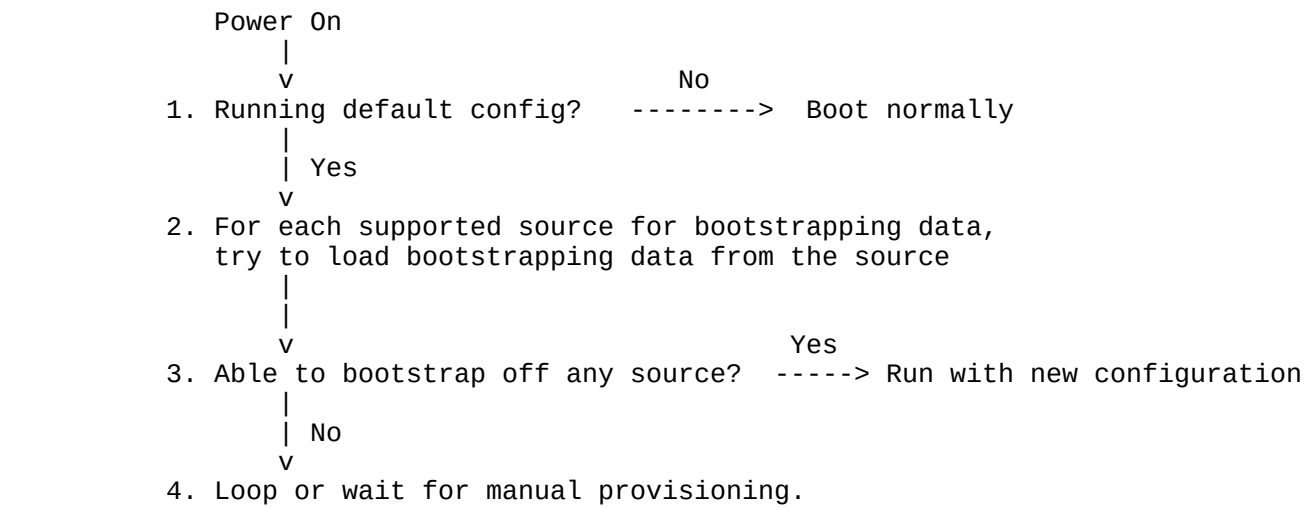
4. Devices that support loading owner signed data (see Section 1.2) MUST also be manufactured with the trust anchor certificate for the device ownership vouchers that the manufacturer provides to prospective owners when it ships out an order of Zero Touch devices (see Section 5.1).

5. Devices MUST be manufactured with an initial device identifier (IDevID), as defined in [Std-802.1AR-2009]. The IDevID is an X.509 certificate, encoding a unique device identifier (e.g., serial number). The device MUST also possess any intermediate certificates between the IDevID certificate and the manufacturer's IDevID trust anchor certificate.

6. Device MUST be manufactured with a private key that corresponds to the public key encoded in the device's IDevID certificate. This private key SHOULD be securely stored, ideally by a cryptographic processor (e.g., a TPM).

6.2. Boot Sequence

A device claiming to support Zero Touch MUST support the boot sequence described in this section.



These interactions are described next.

1. When the device powers on, it first checks to see if it is running the factory default configuration. If it is running a modified configuration, then it 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 6.3.

3. If the device is able to bootstrap itself off any of the sources for bootstrapping data, it runs with the new bootstrapped configuration.

4. Otherwise the device MAY loop back through the list of bootstrapping sources again and/or wait for manual provisioning.

6.3. Processing a Source of Bootstrapping Data

This section describes a recursive algorithm that a device claiming to support Zero Touch MUST use to authenticate bootstrapping data. A device enters this algorithm for each new source of bootstrapping data. The first time the device enters this algorithm, it MUST initialize a conceptual trust state variable, herein referred to as "trust-state", to FALSE. The ultimate goal of this algorithm is for the device to process bootstrap information (not redirect information) while the trust-state variable is TRUE.

If the data source is a bootstrap server, and the device is able to authenticate the server using X.509 certificate path validation ([RFC6125], Section 6) to one of the 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. If trust-state is TRUE, when connecting to the bootstrap server, the device MUST use its IDevID certificate for a client-certificate based authentication and MUST POST progress notifications using the bootstrap server's "notification" action. Otherwise, if trust-state is FALSE, when connecting to the bootstrap server, the device MUST NOT use its IDevID certificate for a client-certificate based authentication and MUST NOT POST progress notifications using the bootstrap server's "notification" action. When accessing a bootstrap server, the device MUST only access its top-level resource, to obtain all the data staged for it in one GET request, so that it can determine if the data is signed or not, and thus act accordingly.

For any data source, if the data is signed (i.e. the data includes a 'signature' field) and the device is able to validate the signed data using the algorithm described in Section 6.4, then the device MUST set trust-state to TRUE, else 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 bootstrap server.

If the data is bootstrap information (not redirect information), and trust-state is FALSE, the device MUST exit the recursive algorithm, returning to the state machine described in Section 6.2. Otherwise, the device MUST attempt to process the bootstrap information as described in Section 6.6. In either case, success of failure, the device MUST exit the recursive algorithm, returning to the state machine described in Section 6.2, the only difference being in how it responds to the "Able to bootstrap off any source?" conditional described in that state machine.

If the data is redirect information, the device MUST process the redirect information as described in Section 6.5. This is the recursion step, it will cause to device to reenter this algorithm, but this time the data source will most definitely be a bootstrap server, as that is all redirect information is able to do, though it's interesting to note that the bootstrap server's response MAY be more redirect information.

6.4. Validating Signed Data

Whenever a device is presented signed data, it MUST validate the signed data as described in this section.

Whenever there is signed data, the device MUST also be provided an ownership voucher and an owner certificate. How all the needed records are provided for each source of bootstrapping data is defined in Section 4

The device MUST first authenticate the ownership voucher by validating the signature on it to one of its preconfigured trust anchors (see Section 6.1) and verify that the voucher contains the device's unique identifier (e.g., serial number). If the authentication of the voucher is successful, the device extracts the Rightful owner's identity from the voucher for use in the next step.

Next the device MUST authenticate the owner certificate by performing X.509 certificate path validation on it to one of its preconfigured trust anchors (see Section 6.1) and by verifying that the Subject contained in the certificate matches the Rightful owner identity extracted from the voucher in the previous step. If the authentication of the certificate is successful, the device extracts the owner's public key from the certificate for use in the next step.

Finally the device MUST authenticate the signed data by verifying the signature on it was generated by the private key matching the public key extracted from the owner certificate in the previous step.

If any of these steps fail, then the device MUST mark the data as invalid and not perform any of the subsequent steps.

6.5. Processing Redirect Information

In order to process redirect information (Section 3.1), the device MUST follow the steps presented in this section.

Processing redirect information is straightforward. Essentially the device MUST immediately attempt to establish a RESTCONF connection to the provided bootstrap server IP address or hostname.

If a hostname is provided, and its DNS resolution is to more than one IP address, the device MUST attempt to try to connect to all of them, sequentially, until it is able to successfully bootstrap off one of them.

If the redirect information includes a trust anchor, and the redirect information can be trusted (e.g., trust-state is TRUE), then the device MUST authenticate the bootstrap server using X.509 certificate path validation ([RFC6125], Section 6) using the specified trust anchor.

6.6. Processing Bootstrap Information

In order to process bootstrap information (Section 3.2), the device MUST follow the steps presented in this section.

When processing bootstrap information, the device MUST first process the boot image information, then commit the initial configuration, and then execute the script, if any, in that order. If the device encounters an error at any step, it MUST NOT proceed to the next step.

First the device MUST determine if the image it is running satisfies the specified "boot-image" criteria. If it does not, the device MUST download, verify, and install the specified boot image, and the reboot. To verify the boot image, the device MUST check that the boot image file matches both the MD5 and SHA fingerprints supplied by the bootstrapping information. Upon rebooting, the device MUST still be in its factory default state, causing the bootstrapping process to run again, which will eventually come to this very point, but this time the device’s running image will satisfy the specified criteria, and thus the device moves to processing the next step.

Next the device commits the provided initial configuration. Assuming no errors, the device moves to processing the next step.

Next, for devices that support executing scripts, if a script has been specified, the device executes the script, checking its exit status code to determine if it succeeded, had warning, or had errors. In the case of errors, the device MUST reset itself in such a way that force the reinstallation of its boot image, thereby wiping out any bad state the script might have left behind.

At this point, the device has completely processed the bootstrapping data and is ready to be managed. If the configuration configured the device it initiate a call home connection, it should proceed to do so now. Otherwise, the device should wait for a NETCONF or RESTCONF client to connect to it.

7. YANG-defined API and Artifacts

Central to the solution presented in this document is the use of a YANG module [RFC6020] to simultaneously define a RESTCONF based API for a bootstrap/redirect server as well as the encoding for signed artifacts that can be conveyed outside of the RESTCONF protocol (DHCP, FTP, TFTP, etc.).

The module defined in this section makes extensive use of data types defined in [RFC2315], [RFC5280], [RFC6991], and [RFC5280].

7.1. Module Overview

The following tree diagram Section 1.3 provides an overview for both the API and artifacts that can be used outside of RESTCONF.

```
module: ietf-zerotouch-bootstrap-server
   +--ro devices
      +--ro device* [unique-id]
         +--ro unique-id                string
         +--ro (type)?
         |  +--:(redirect-information)
         |  |  +--ro redirect-information
         |  |     +--ro bootstrap-server* [address]
         |  |        +--ro address         inet:host
         |  |        +--ro port?           inet:port-number
         |  |        +--ro trust-anchor    binary
         |  +--:(bootstrap-information)
         |     +--ro bootstrap-information
         |        +--ro boot-image
         |        |  +--ro modules
         |        |  |  +--ro module*
         |        |  |     +--ro name?       yang:yang-identifier
         |        |  |     +--ro revision?   string
         |        |  +--ro name       string
         |        |  +--ro md5        string
         |        |  +--ro sha1       string
         |        |  +--ro uri*       inet:uri
         |        +--ro configuration
         |        +--ro script?          string
         +--ro owner-certificate
         |  +--ro certificate    binary
         |  +--ro issuer-crl?    binary
         +--ro ownership-voucher
         |  +--ro voucher       binary
         |  +--ro issuer-vrl?   binary
         +--ro signature?               binary
         +---x notification
            +---w input
               +---w notification-type    enumeration
               +---w message?             string
               +---w ssh-host-keys
               |  +---w ssh-host-key*
               |     +---w format      enumeration
               |     +---w key-data    string
               +---w trust-anchors
                  +---w trust-anchor*
                     +---w protocol*      enumeration
                     +---w certificate    binary
```

In the above diagram, notice that all of the protocol accessible node are read-only, to assert that devices can only pull data from the bootstrap server.

Also notice that the module defines an action statement, which devices may use to provide progress notifications to the bootstrap server.

7.2. API Examples

This section presents some examples illustrating device interactions with a bootstrap server to access Redirect and Bootstrap information, both unsigned and signed, as well as to send a progress notification. These examples show the bootstrap information containing configuration defined by [RFC7317] and [draft-ietf-netconf-server-model].

7.2.1. Unsigned Redirect Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives unsigned redirect information. This example is representative of a response a trusted redirect server might return.

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

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml


RESPONSE
--------

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<!-- '\' line wrapping added for formatting purposes only -->

<device
   xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <redirect-information>
```

```
     <bootstrap-server>
       <address>phs1.example.com</address>
       <port>8443</port>
       <trust-anchor>
         WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
         lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
         zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
         NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
         VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
         V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
         NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
         Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
         WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
         QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
         MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
         25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
         RJSUJQFRStS0Cg==
       </trust-anchor>
     </bootstrap-server>
     <bootstrap-server>
       <address>phs2.example.com</address>
       <port>8443</port>
       <trust-anchor>
         WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
         lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
         zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
         NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
         VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
         V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
         NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
         Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
         WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
         QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
         MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
         25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
         RJSUJQFRStS0Cg==
       </trust-anchor>
     </bootstrap-server>
   </redirect-information>
 </device>
```

7.2.2. Signed Redirect Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives signed redirect information. This example is representative of a response that redirect server might return if concerned the device might not be able to authenticate its TLS certificate.

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

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml


RESPONSE
--------

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<!-- '\' line wrapping added for formatting purposes only -->

<device
   xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <redirect-information>
    <bootstrap-server>
      <address>phs1.example.com</address>
      <port>8443</port>
      <trust-anchor>
        WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
        lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
        zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
        NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
        VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
        V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
        NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
        Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
        WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
        QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
        MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
        25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
        RJSUJQFRStS0Cg==
      </trust-anchor>
    </bootstrap-server>
    <bootstrap-server>
      <address>phs2.example.com</address>
      <port>8443</port>
      <trust-anchor>
        WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
```

```
        lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
        zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
        NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
        VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
        V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
        NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
        Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
        WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
        QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
        MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
        25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
        RJSUJQFRStS0Cg==
      </trust-anchor>
    </bootstrap-server>
  </redirect-information>
  <owner-certificate>
    <certificate>
      MIIExTCCA62gAwIBAgIBATANBgkqhkiG9w0BAQsFADCBqjELMAkGA1UEBhMCVVMx\
      EzARBgNVBAgTCkNhbGlmb3JuaWExEjAQBgNVBAcTCVN1bm55dmFsZTEZMBcGA1UE\
      ChQQSnVuaXBlcl9OZXR3b3JrczEdMBsGA1UECxQUQ2VydGlmaWNhdGVfSXNzdWFu\
      Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
      QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
      RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
      ap1DgmS3IaYl/s4OOF8yzcYJprm8O7NyZp+Y9H1U/7Qfp97/KbqwCgkHSzOlnt0X\
      KQTpIM/rNrbrkuTmalezFoFS7mrxLXJAsfP1guVcD7sLCyjvegL8pRCCrU9xyKLF\
      8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
      KmORbiKU2GTGZkaCgCjmrWpvrYWLoXv/sf2nPLyK6YjiWsslOJtRO+KzRbs2B18C\
      AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
      yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
      GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
      X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
      MjAOBgNVHQ8BAf8EBAMCAgQwQgYDVR0fBDswOTA3oDWgM4YxaHR0cDovL2NybC5q\
      dW5pcGVyLm5ldD9jYT1KdW5pcGVyX1RydXN0X0FuY2hvcl9DQTANBgkqhkiG9w0B\
      AQsFAAOCAQEAOuD7EBilqQcT3t2C4AXta1gGNNwdldLLw0jtk4BMiA9l//DZfskB\
      2AaJtiseLTXsMF6MQwDs1YKkiXKLu7gBZDlJ6NiDwy1UnXhi2BDG+MYXQrc6p76K\
      z3bsVwZlaJQCdF5sbggc1MyrsOu9QirnRZkIv3R8ndJH5K792ztLquulAcMfnK1Y\
      NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX7WJzEbT/G7MUfo\
      Sb+U2PVsQTDWEzUjVnG7vNWYxirnAOZ0OXEWWYxHUJntx6DsbXYuX7D1PkkNr7ir\
      96DpOPtX7h8pxxGSDPBXIyvg02aFMphstQ==
    </certificate>
    <issuer-crl>
      Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
```

```
      yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
      GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
      X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
      MjAO==
    </issuer-crl>
  </owner-certificate>
  <ownership-voucher>
    <voucher>
      ChQQSnVuaXBlcl9OZXR3b3JrczEdMBsGA1UECxQUQ2VydGlmaWNhdGVfSXNzdWFu\
      Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
      yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
      GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
      X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
      MjAO
    </voucher>
    <issuer-crl>
      QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
      RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
      KQTpIM/rNrbrkuTmalezFoFS7mrxLXJAsfP1guVcD7sLCyjvegL8pRCCrU9xyKLF\
      8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
      AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
    </issuer-crl>
  </ownership-voucher>
  <signature>
    RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
    QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
    MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
    NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
  </signature>
</device>
```

7.2.3. Unsigned Bootstrap Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives unsigned bootstrapping information. This example is representative of a response a locally deployed bootstrap server might return.

REQUEST

['\' line wrapping added for formatting only]

```
GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml
```

RESPONSE

```
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<!-- '\' line wrapping added for formatting purposes only -->

<device
   xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <bootstrap-information>
    <boot-image>
      <name>
        boot-image-v3.2R1.6.img
      </name>
      <md5>
        SomeMD5String
      </md5>
      <sha1>
        SomeSha1String
      </sha1>
      <uri>
        ftp://ftp.example.com/path/to/file
      </uri>
    </boot-image>
    <configuration>
      <!-- from ietf-system.yang -->
      <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
        <authentication>
          <user>
            <name>admin</name>
            <ssh-key>
              <name>admin's rsa ssh host-key</name>
              <algorithm>ssh-rsa</algorithm>
              <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsR\
```

```
              jCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2Mw\
              E1lG9YxLzeS5p2ngzK61vikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVc\
              WAw1lOr9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA\
              vg7SLqQFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jW\
              EIuA7LvEJYql4unq4Iog+/+CiumTkmQIWRgIoj4FCzYkO9NvRE6fOSLLf\
              gakWVOZZgQ8929uWjCWlGlqn2mPibp2Go1</key-data>
            </ssh-key>
          </user>
        </authentication>
      </system>
      <!-- from ietf-netconf-server.yang -->
      <netconf-server
        xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
        <call-home>
          <application>
            <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>my-call-home-x509-key</host-key>
              </host-keys>
            </ssh>
          </application>
        </call-home>
      </netconf-server>
    </configuration>
  </bootstrap-information>
</device>
```

7.2.4. Signed Bootstrap Information

The following example illustrates a device using the API to fetch its bootstrapping data. In this example, the device receives signed bootstrap information. This example is representative of a response that bootstrap server might return if concerned the device might not be able to authenticate its TLS certificate.

REQUEST

['\' line wrapping added for formatting only]

```
GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
devices/device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml


RESPONSE
--------

HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml

<!-- '\' line wrapping added for formatting purposes only -->

<device
   xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <bootstrap-information>
    <boot-image>
      <name>
        boot-image-v3.2R1.6.img
      </name>
      <md5>
        SomeMD5String
      </md5>
      <sha1>
        SomeSha1String
      </sha1>
      <uri>
        /path/to/on/same/bootserver
      </uri>
    </boot-image>
    <configuration>
      <!-- from ietf-system.yang -->
      <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
        <authentication>
          <user>
            <name>admin</name>
            <ssh-key>
              <name>admin's rsa ssh host-key</name>
              <algorithm>ssh-rsa</algorithm>
              <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsR\
              jCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2Mw\
              E1lG9YxLzeS5p2ngzK61vikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVc\
```

```
              WAw1lOr9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA\
              vg7SLqQFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jW\
              EIuA7LvEJYql4unq4Iog+/+CiumTkmQIWRgIoj4FCzYkO9NvRE6fOSLLf\
              gakWVOZZgQ8929uWjCWlGlqn2mPibp2Go1</key-data>
            </ssh-key>
          </user>
        </authentication>
      </system>
      <!-- from ietf-netconf-server.yang -->
      <netconf-server
        xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
        <call-home>
          <application>
            <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>my-call-home-x509-key</host-key>
              </host-keys>
            </ssh>
          </application>
        </call-home>
      </netconf-server>
    </configuration>
  </bootstrap-information>
  <owner-certificate>
    <certificate>
      MIIExTCCA62gAwIBAgIBATANBgkqhkiG9w0BAQsFADCBqjELMAkGA1UEBhMCVVMx\
      EzARBgNVBAgTCkNhbGlmb3JuaWExEjAQBgNVBAcTCVN1bm55dmFsZTEZMBcGA1UE\
      ChQQSnVuaXBlcl9OZXR3b3JrczEdMBsGA1UECxQUQ2VydGlmaWNhdGVfSXNzdWFu\
      Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
      QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
      RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
      ap1DgmS3IaYl/s4OOF8yzcYJprm8O7NyZp+Y9H1U/7Qfp97/KbqwCgkHSzOlnt0X\
      KQTpIM/rNrbrkuTmalezFoFS7mrxLXJAsfP1guVcD7sLCyjvegL8pRCCrU9xyKLF\
      8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
      KmORbiKU2GTGZkaCgCjmrWpvrYWLoXv/sf2nPLyK6YjiWsslOJtRO+KzRbs2B18C\
```

```
      AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
      yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
      GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
      X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
      MjAOBgNVHQ8BAf8EBAMCAgQwQgYDVR0fBDswOTA3oDWgM4YxaHR0cDovL2NybC5q\
      dW5pcGVyLm5ldD9jYT1KdW5pcGVyX1RydXN0X0FuY2hvcl9DQTANBgkqhkiG9w0B\
      AQsFAAOCAQEAOuD7EBilqQcT3t2C4AXta1gGNNwdldLLw0jtk4BMiA9l//DZfskB\
      2AaJtiseLTXsMF6MQwDs1YKkiXKLu7gBZDlJ6NiDwy1UnXhi2BDG+MYXQrc6p76K\
      z3bsVwZlaJQCdF5sbggc1MyrsOu9QirnRZkIv3R8ndJH5K792ztLquulAcMfnK1Y\
      NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX7WJzEbT/G7MUfo\
      Sb+U2PVsQTDWEzUjVnG7vNWYxirnAOZ0OXEWWYxHUJntx6DsbXYuX7D1PkkNr7ir\
      96DpOPtX7h8pxxGSDPBXIyvg02aFMphstQ==
    </certificate>
    <issuer-crl>
      Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
      yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
      GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
      X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
      MjAO==
    </issuer-crl>
  </owner-certificate>
  <ownership-voucher>
    <voucher>
      ChQQSnVuaXBlcl9OZXR3b3JrczEdMBsGA1UECxQUQ2VydGlmaWNhdGVfSXNzdWFu\
      Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
      yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
      GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
      X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
      MjAO
    </voucher>
    <issuer-vrl>
      QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
      MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
      ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
      RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
      KQTpIM/rNrbrkuTmalezFoFS7mrxLXJAsfP1guVcD7sLCyjvegL8pRCCrU9xyKLF\
      8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
      AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
```

```
      WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
      NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
    </issuer-vrl>
  </ownership-voucher>
  <signature>
    RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
    QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
    MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
    NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
  </signature>
</device>
```

7.2.5. Progress Notifications

The following example illustrates a device using the API to post a notification to the server. The device may send more than one notification to the server (e.g., to provide status updates).

The bootstrap server MUST NOT process a notification from a device without first authenticating the device. This is in contrast to when a device is fetching data from the server, a read-only operation, in which case device authentication is not strictly required.

In this example, the device sends a notification indicating that it has completed bootstrapping off the data provided by the server. This example also illustrates the device sending its SSH host keys to the bootstrap server, which it might, for example, forward onto a downstream NMS component, so that the NMS can subsequently authenticate the device when establishing a NETCONF over SSH connection to it.

Note that the need for a device to provide its SSH host key (or TLS server certificate) in the "bootstrap-complete" message is unnecessary when the device is able to present its IDevID certificate [Std-802.1AR-2009] as its SSH host key or TLS server certificate, when subsequently establishing a NETCONF or RESTCONF connection with the deployment-specific NMS.

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

POST https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
devices/device=123456/notification HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml

<!-- '\' line wrapping added for formatting purposes only -->
```

```
<input
   xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <notification-type>bootstrap-complete</notification-type>
  <message>example message</message>
  <ssh-host-keys>
    <ssh-host-key>
      <format>ssh-rsa</format>
      <key-data>
       AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRCjCzfve2m6\
       zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2MwjE1lG9YxL\
       zeS5p2ngzK61vikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVcCWAw1lOr\
       9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA5vg7SLq\
       QFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jWqEIuA7\
       LvEJYql4unq4Iog+/+CiumTkmQIWRgIoj4FCzYkO9NvRE6fOSLLf6gakW\
       VOZZgQ8929uWjCWlGlqn2mPibp2Go1
      </key-data>
    </ssh-host-key>
    <ssh-host-key>
      <format>ssh-dsa</format>
      <key-data>
       zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2MwjE1lG9YxL\
       zeS5p2ngzK61vikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVcCWAw1lOr\
       9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA5vg7SLq\
       AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRCjCzfve2m6\
       QFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jWqEIuA7\
       LvEJYql4unq4Iog+/+CiumTkmQIWRgIoj4FCzYkO9NvRE6fOSLLf6gakW\
       VOZZgQ8929uWjCWlGlqn2mPibp2Go1
      </key-data>
    </ssh-host-key>
  </ssh-host-keys>
  <trust-anchors>
    <trust-anchor>
      <protocol>netconf-ssh</protocol>
      <protocol>netconf-tls</protocol>
      <protocol>restconf-tls</protocol>
      <protocol>netconf-ch-ssh</protocol>
      <protocol>netconf-ch-tls</protocol>
      <protocol>restconf-ch-tls</protocol>
      <certificate>
        WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
        lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
        zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
        NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
        VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
        V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
        NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
        Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
        WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
```

```
        QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
        MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
        25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
        RJSUJQFRStS0Cg==
      </certificate>
    </trust-anchor>
  </trust-anchors>

</input>
```

```
RESPONSE
--------

HTTP/1.1 204 No Content
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
```

7.3. Artifact Examples

This section presents some examples for how the same information provided by the API can be packaged into stand alone artifacts. The encoding for these artifacts is the same as if an HTTP GET request had been sent to the RESTCONF URL for the specific resource. These examples show the bootstrap information containing configuration defined by [RFC7317] and [draft-ietf-netconf-server-model].

Encoding these artifacts for use outside of the RESTCONF protocol extends their utility for other deployment scenarios, such as when a local DHCP server or a removable storage device is used. By way of example, this may be done to address an inability for the device to access an Internet facing bootstrap/redirect server, or just for a preference to use locally deployed infrastructure.

7.3.1. Signed Redirect Information

The following example illustrates how a redirect can be encoded into an artifact for use outside of the RESTCONF protocol. The redirect information is signed so that it is secure even when no transport-level security is provided.

```
 <!-- '\' line wrapping added for formatting purposes only -->

 <redirect-information
    xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
   <bootstrap-server>
     <address>phs1.example.com</address>
     <port>8443</port>
     <trust-anchor>
       WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
       lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
       zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
       NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
       VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
       V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
       NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
       Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
       WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
       QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
       MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
       25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
       RJSUJQFRStS0Cg==
     </trust-anchor>
   </bootstrap-server>
   <bootstrap-server>
     <address>phs1.example.com</address>
     <port>8443</port>
     <trust-anchor>
       WmdsK2gyTTg3QmtGMjhWbW1CdFFVaWc3OEgrRkYyRTFwdSt4ZVRJbVFFM\
       lLQllsdWpOcjFTMnRLR05EMUc2OVJpK2FWNGw2NTdZNCtadVJMZgpRYjk\
       zSFNwSDdwVXBCYnA4dmtNanFtZjJma3RqZHBxeFppUUtTbndWZTF2Zwot\
       NGcEk3UE90cnNFVjRwTUNBd0VBQWFPQ0FSSXdnZ0VPCk1CMEdBMVVkRGd\
       VEJiZ0JTWEdlbUEKMnhpRHVOTVkvVHFLNWd4cFJBZ1ZOYUU0cERZd05ER\
       V6QVJCZ05WQkFNVENrTlNUQ0JKYzNOMVpYS0NDUUNVRHBNSll6UG8zREF\
       NQmdOVkhSTUJBZjhFCkFqQUFNQTRHQTFVZER3RUIvd1FFQXdJSGdEQnBC\
       Z05WSFI4RVlqQmdNRjZnSXFBZ2hoNW9kSFJ3T2k4dlpYaGgKYlhCc1pTN\
       WpiMjB2WlhoaGJYQnNaUzVqY215aU9LUTJNRFF4Q3pBSkJnTlZCQVlUQW\
       QmdOVkJBWVRBbFZUTVJBd0RnWURWUVFLRXdkbAplR0Z0Y0d4bE1RNHdEQ\
       MkF6a3hqUDlVQWtHR0dvS1U1eUc1SVR0Wm0vK3B0R2FieXVDMjBRd2kvZ\
       25PZnpZNEhONApXY0pTaUpZK2xtYWs3RTRORUZXZS9RdGp4NUlXZmdvN2\
   </bootstrap-server>
   <signature>
     RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
     QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
     MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
     NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
   </signature>
 </redirect-information>
```

7.3.2. Signed Bootstrap Information

The following example illustrates how bootstrapping data can be encoded into an artifact for use outside of the RESTCONF protocol. The bootstrap information is signed so that it is secure when no transport-level security is provided.

```
<!-- '\' line wrapping added for formatting purposes only -->

<bootstrap-information
   xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
  <boot-image>
    <name>
      boot-image-v3.2R1.6.img
    </name>
    <md5>
      SomeMD5String
    </md5>
    <sha1>
      SomeSha1String
    </sha1>
    <uri>
      file:///some/path/to/raw/file
    </uri>
  </boot-image>
  <configuration>
    <!-- from ietf-system.yang -->
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      <authentication>
        <user>
          <name>admin</name>
          <ssh-key>
            <name>admin's rsa ssh host-key</name>
            <algorithm>ssh-rsa</algorithm>
            <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRC\
            jCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2Mwj\
            E1lG9YxLzeS5p2ngzK61vikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVcC\
            WAw1lOr9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA5\
            vg7SLqQFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jWq\
            EIuA7LvEJYql4unq4Iog+/+CiumTkmQIWRgIoj4FCzYkO9NvRE6fOSLLf6\
            gakWVOZZgQ8929uWjCWlGlqn2mPibp2Go1</key-data>
          </ssh-key>
        </user>
      </authentication>
    </system>
    <!-- from ietf-netconf-server.yang -->
    <netconf-server
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
```

```
       <call-home>
         <application>
           <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>my-call-home-x509-key</host-key>
             </host-keys>
           </ssh>
         </application>
       </call-home>
     </netconf-server>
   </configuration>
 </bootstrap-information>
```

7.3.3. Owner Certificate

The following example illustrates how the owner certificate, along with its CRL, can be encoded into an artifact for use outside of the RESTCONF protocol. Note that the inclusion of the CLR is optional, and only present to support cases where the device is deployed on a private network, such that it would be unable to validate the revocation status of the certificate using an online lookup of the CRL or using OCSP. As the owner certificate and CRL are already signed by the manufacturer, an additional owner signature is unnecessary.

```
 <!-- '\' line wrapping added for formatting purposes only -->

 <owner-certificate
    xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
   <certificate>
     MIIExTCCA62gAwIBAgIBATANBgkqhkiG9w0BAQsFADCBqjELMAkGA1UEBhMCVVMx\
     EzARBgNVBAgTCkNhbGlmb3JuaWExEjAQBgNVBAcTCVN1bm55dmFsZTEZMBcGA1UE\
     ChQQSnVuaXBlcl9OZXR3b3JrczEdMBsGA1UECxQUQ2VydGlmaWNhdGVfSXNzdWFu\
     Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
     QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
     MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
     ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
     RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
     ap1DgmS3IaYl/s4OOF8yzcYJprm8O7NyZp+Y9H1U/7Qfp97/KbqwCgkHSzOlnt0X\
     KQTpIM/rNrbrkuTmalezFoFS7mrxLXJAsfP1guVcD7sLCyjvegL8pRCCrU9xyKLF\
     8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
     KmORbiKU2GTGZkaCgCjmrWpvrYWLoXv/sf2nPLyK6YjiWsslOJtRO+KzRbs2B18C\
     AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
     yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
     WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
     MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
     GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
     X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
     MjAOBgNVHQ8BAf8EBAMCAgQwQgYDVR0fBDswOTA3oDWgM4YxaHR0cDovL2NybC5q\
     dW5pcGVyLm5ldD9jYT1KdW5pcGVyX1RydXN0X0FuY2hvcl9DQTANBgkqhkiG9w0B\
     AQsFAAOCAQEAOuD7EBilqQcT3t2C4AXta1gGNNwdldLLw0jtk4BMiA9l//DZfskB\
     2AaJtiseLTXsMF6MQwDs1YKkiXKLu7gBZDlJ6NiDwy1UnXhi2BDG+MYXQrc6p76K\
     z3bsVwZlaJQCdF5sbggc1MyrsOu9QirnRZkIv3R8ndJH5K792ztLquulAcMfnK1Y\
     NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX7WJzEbT/G7MUfo\
     Sb+U2PVsQTDWEzUjVnG7vNWYxirnAOZ0OXEWWYxHUJntx6DsbXYuX7D1PkkNr7ir\
     96DpOPtX7h8pxxGSDPBXIyvg02aFMphstQ==
   </certificate>
   <issuer-crl>
     Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
     MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
     ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
     yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
     WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
     MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
     GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
     X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
     MjAO==
   </issuer-crl>
 </owner-certificate>
```

7.3.4. Ownership Voucher

The following example illustrates how the ownership voucher, along with its CRL, can be encoded into an artifact for use outside of the RESTCONF protocol. Note that the inclusion of the CLR is optional, and only present to support cases where the device is deployed on a private network, such that it would be unable to validate the revocation status of the certificate using an online lookup of the CRL or using OCSP. As the ownership voucher and CRL are already signed by the manufacturer, an additional owner signature is unnecessary.

```
 <!-- '\' line wrapping added for formatting purposes only -->

 <ownership-voucher
    xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server">
   <voucher>
     ChQQSnVuaXBlcl9OZXR3b3JrczEdMBsGA1UECxQUQ2VydGlmaWNhdGVfSXNzdWFu\
     Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh\
     MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
     ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
     yh/JaftWYf7m3KBzOdg2MIHfBgNVHSMEgdcwgdSAFDSljCNmTN5b+CDujJLlyDal\
     WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
     MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w\
     GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0\
     X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQDUbsEdTn5v\
     MjAO
   </voucher>
   <issuer-crl>
     QGp1bmlwZXIuY29tMB4XDTE0MDIyNzE0MTM1MloXDTE1MDIyNzE0MTM1MlowMDET\
     MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC\
     ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANL5Mk5qFsVuqo+JmXWLmFxI\
     RDEuRiZNRNLeJpgN9YWkXLAZX2rASwy041EMmZ6KAkWUd3ZmXucfoLpdRemfuPii\
     KQTpIM/rNrbrkuTmalezFoFS7mrxLXJAsfP1guVcD7sLCyjvegL8pRCCrU9xyKLF\
     8u/Qz4s0x0uzcGYh0sd3iWj21+AtigSLdMD76/j/VzftQL8B1yp3vc1EZiowOwq4\
     AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFHppoyXF\
     WFPaoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES\
     NTOufhQsD2t4TYpEkzLEiZqSswdBOaPxPcJLQNW8Bw2xN+A9GX=
   </issuer-crl>
 </ownership-voucher>
```

7.4. YANG Module

The bootstrap server's device-facing interface is normatively defined by the following YANG module:

<CODE BEGINS> file "ietf-zerotouch-bootstrap-server@2016-04-06.yang"

```
module ietf-zerotouch-bootstrap-server {
  yang-version "1.1";

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server";
  prefix "ztbs";

  import ietf-yang-types {     // RFC 6991
    prefix yang;
  }

  import ietf-inet-types {     // RFC 6991
    prefix inet;
  }

  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 the southbound interface for Zero Touch
    bootstrap 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 XXXX; see
    the RFC itself for full legal notices.";

  revision "2016-04-06" {
    description
     "Initial version";
```

```
    reference
     "RFC XXXX: Zero Touch Provisioning for NETCONF Call Home";
  }


  container devices {
    config false;
    description
      "This is the top-level container for a device-facing protocol.
       As such it is read-only, how this data is configured is outside
       the scope of this data-model.  Further, it is expected that
       devices would only be able to access their data and not the
       data for any other device.";
    list device {
      key unique-id;

      description
        "A device's record entry.  This is the only RESTCONF resource
         that a device is expected to GET.  Getting this just this
         top-level provides the device with all the data it needs in
         a single request, which is ideal from both a performance and
         a resiliency perspectives..";

      leaf unique-id {
        type string;
        description
          "A unique identifier for the device (e.g., serial number).
           Each device accesses its bootstrapping record by its unique
           identifier.";
      }

      choice type {
        description
          "This choice statement ensures the response only contains
           redirect-information or bootstrap-information.";

        container redirect-information {
          description
            "This is redirect information data.  Its purpose is to
             redirect the device to another bootstrap server.  It
             contains a list of bootstrap servers.";

          list bootstrap-server {
            key address;
            description
              "A bootstrap server entry.";

            leaf address {
```

```
            type inet:host;
            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.";
          }
          leaf trust-anchor {
            type binary;
            mandatory true;
            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.  A
               certificate that a device can use as a trust anchor to
               authenticate the bootstrap server it is being redirected
               to.";
            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).";
          }
        }
      }

      container bootstrap-information {
        description
          "This is bootstrap information data.  Its purpose is to
           provide the device everything it needs to bootstrap
           itself.";

        container boot-image {
          description
            "Specifies criteria for the boot image the device MUST be
             running.";

          container modules {
            description
              "Specifies a list of YANG modules that the device MUST
```

```
          support.  This node is optional.  When this node is
          specified, the remaining nodes MUST be processed only
          in case the currently running image does not support
          any of the YANG modules, as a means to obtain a valid
          image.  When this node is not specified, then the
          device MUST ensure it is running the exact image, as
          specified by the remaining 'boot-image' nodes.";
        list module {
          description
            "Specifies a specific YANG modules, by its name and
             revision date.  The revision date is provided as a
             minimal revision date, and supported revision
             thereafter is considered sufficient";
          leaf name {
            type yang:yang-identifier;
            description
              "The YANG module's name.";
          }
          leaf revision {
            type string {
              pattern '\d{4}-\d{2}-\d{2}';
            }
            description
              "Represents a specific date in 2016-04-06 format.";
          }
        }
      }
      leaf name {
        type string;
        mandatory true;
        description
          "The name of a software image that either the device
           MUST be running, or MUST install only if its currently
           running image cannot support any of the required YANG
           modules.";
      }
      leaf md5 {
        type string;
        mandatory true;
        description
          "The hex-encoded MD5 hash over the boot-image file.";
      }
      leaf sha1 {
        type string;
        mandatory true;
        description
          "The hex-encoded SHA-1 hash over the boot-image file.";
      }
```

```
            leaf-list uri {
              type inet:uri;
              min-elements 1;
              description
                "An ordered list of URIs to where the boot-image file
                 may be obtained.  When the bootstrap information is
                 obtained from a bootstrap server, it is RECOMMENDED
                 that the list begins with absolute paths (e.g.,
                 beginning with '/') to the bootstrap server, so as
                 to leverage the existing secure connection.  If remote
                 URLs are also present in the list, deployments MUST
                 know in advance which URI schemes (https, http, ftp,
                 file, etc.) a device supports.  If a secure scheme
                 (e.g., https) is provided, devices MAY blindly accept
                 the server's credentials (e.g., TLS certificate).
                 Regardless how obtained, the device MUST ensure that
                 the boot-image is valid, either by leveraging a
                 signature embedded in the boot-image itself, if it
                 exists, or by first comparing the downloaded image to
                 both the MD5 and SHA1 fingerprints provided above.";
            }
          }

          anyxml configuration {   // pyang doesn't support anydata yet!
            description
              "Any configuration data model known to the device.  It may
               contain manufacturer-specific and/or standards-based data
               models.";
          }

          leaf script {
            type string;
            description
              "A device specific script that enables the execution of
               commands to perform actions not possible thru configuration
               alone.  The script SHOULD be executed with 'root' level
               permissions.

               If a script is erroneously provided to a device that
               does not support the execution of scripts, the device
               SHOULD send a 'script-warning' notification message,
               but otherwise continue processing the bootstrapping
               data as if the script had not been present.

               The script would return exit status code '0' on success
               and non-zero on error, with accompanying stderr/stdout
               for logging purposes.  In the case of an error, the exit
               status code will specify what the device should do.
```

```
            If the exit status code is greater than zero, then the
            device should assume that the script had soft failure
            that the script believes does not affect manageability.
            If the device obtained the bootstrap information from
            a bootstrap server, it SHOULD send a 'script-warning'
            notification message.

            If the exit status code is less than zero, the device
            should assume the script had a hard error that the
            script believes will affect manageability.  In this
            case, the device should try to send a 'script-error'
            notification message followed by a reset that will
            force a new boot-image install (wiping out anything
            the script may have done) and restart the entire
            bootstrapping process again.";
        }
      }
    }

    container owner-certificate {
      when "../ownership-voucher" {
        description
          "The owner certificate is only configurable when there
           also exists an ownership voucher.";
      }
      description
        "It is intended that the device will fetch this container
         as a whole, as it contains values that need to be
         processed together.";

      leaf certificate {
        type binary;
        mandatory true;
        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.
           This certificate, signed by a manufacturer or delegate,
           for an owner, must encode a manufacturer-assigned value
           identifying the organization. This identifier must match
           the owner identifier encoded in the ownership voucher.";
        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).";
        }

        leaf issuer-crl {
          type binary;
          description
            "An CRL structure as specified by RFC 5280, Section 5
             encoded using the ASN.1 distinguished encoding rules
             (DER), as specified in ITU-T X.690.  The CRL for the
             CA that signed the owner certificate.  The CRL should
             be as up to date as possible.  This leaf is optional
             as it is only needed to support deployments where the
             device is unable to download the CRL from and of the
             distribution points listed in the owner certificate.";
          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).";
          }
        }

        container ownership-voucher {
          when "../signature" {
            description
              "An ownership voucher is only configurable when there
               also exists a signature.";
          }
          must "../owner-certificate" {
            description
              "An owner certificate must be present whenever an
               ownership voucher is present.";
          }
          description
            "This container contains the ownership voucher that the
             device uses to ascertain the identity of its rightful
             owner, as certified by its manufacturer.";

          leaf voucher {
            type binary;
            mandatory true;
            description
              "A manufacturer-specific encoding binding unique device
```

```
            identifiers to an owner identifier value matching the
            value encoded in the owner-certificate below.";
        }

        leaf issuer-vrl {
          type binary;
          description
            "An manufacturer-specific encoding of a voucher revocation
             list (VRL) for the issuer used by the manufacturer or
             delegate to sign ownership vouchers.  The VRL should be
             as up to date as possible.  This leaf is optional as it
             is only needed to support deployments where the device
             is unable to download the VRL from the manufacturer or
             delegate using some manufacturer-specific mechanism.";
        }
      }

      leaf signature {
        type binary;
        must "../ownership-voucher" {
          description
            "An ownership voucher must be present whenever an
             signature is present.";
        }
        description
          "A PKCS #7 SignedData structure as specified by RFC
           2315, Section 9.1 encoded using the ASN.1 distinguished
           encoding rules (DER), as specified in ITU-T X.690.
           This signature is generated using the owner's private
           private key and an owner-selected digest algorithm over
           the redirect-information or the bootstrap-information
           nodes, which ever is present, and in whatever encoding
           they are presented in (e.g., XML, JSON, etc.).";
           // is there a canonical format?
        reference
          "RFC 2315:
              PKCS #7: Cryptographic Message Syntax Version 1.5
           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 notification {
        input {
          leaf notification-type {
```

```
            type enumeration {
              enum bootstrap-initiated {
                description
                  "Indicates that the device has just accessed
                   the bootstrap server.  The 'message' field
                   below SHOULD contain any additional information
                   that the manufacturer thinks might be useful,
                   or omitted entirely.";
              }
              enum validation-error {
                description
                  "Indicates that the device had an issue validating
                   the response from the bootstrap server.  The
                   'message' field below SHOULD indicate the specific
                   error.  This message also indicates that the device
                   has abandoned trying to bootstrap off this bootstrap
                   server.";
              }
              enum signature-validation-error {
                description
                  "Indicates that the device had an issue validating
                   the bootstrapping data.  For instance, this could
                   be due to the device expecting signed data, but
                   only found unsigned data, or because the ownership
                   voucher didn't include its unique identifier, or
                   because the signature didn't match, or and other
                   relevant error.  This 'message' field below SHOULD
                   indicate the specific error.  This message also
                   indicates that the device has abandoned trying to
                   bootstrap off this bootstrap server.";
              }
              enum image-mismatch {
                description
                  "Indicates that the device has determined that
                   its running image does not meet the specified
                   criteria.  The 'message' field below SHOULD
                   indicate both what image the device is currently
                   running as well as the criteria that failed.";
              }
              enum image-download-error {
                description
                  "Indicates that the device had an issue downloading
                   the image, which could be anything from the file
                   server being unreachable to the downloaded file
                   being the incorrect file (signature mismatch). The
                   'message' field about SHOULD indicate the specific
                   error.  This message also indicates that the device
                   has abandoned trying to bootstrap off this bootstrap
```

```
          server.";
        }
        enum config-warning {
          description
            "Indicates that the device obtained warning messages
             when it committed the initial configuration.  The
             'message' field below SHOULD indicate the warning
             messages that were generated.";
        }
        enum config-error {
          description
            "Indicates that the device obtained error messages
             when it committed the initial configuration.  The
             'message' field below SHOULD indicate the error
             messages that were generated.  This message also
             indicates that the device has abandoned trying to
             bootstrap off this bootstrap server.";
        }
        enum script-warning {
          description
            "Indicates that the device obtained a greater than
             zero exit status code from the script when it was
             executed.  The 'message' field below SHOULD indicate
             both the resulting exit status code and well as
             capture any stdout/stderr messages the script may
             have produced.";
        }
        enum script-error {
          description
            "Indicates that the device obtained a less than zero
             exit status code from the script when it was executed.
             The 'message' field below SHOULD indicate both the
             resulting exit status code and well as capture any
             stdout/stderr messages the script may have produced.
             This message also indicates that the device has
             abandoned trying to bootstrap off this bootstrap
             server.";
        }
        enum bootstrap-complete {
          description
            "Indicates that the device successfully processed the
             all the bootstrapping data and that it is ready to
             be managed.  The 'message' field below SHOULD contain
             any additional information that the manufacturer
             thinks might be useful, or omitted entirely.  At
             this point, the device is not expected to access
             the bootstrap server again.";
        }
```

```
            enum informational {
              description
                "Provided any additional information not captured by
                 any of the other notification-type.  The 'message'
                 field below SHOULD contain any additional information
                 that the manufacturer thinks might be useful, or
                 omitted entirely.";
            }
          }
          mandatory true;
          description
            "The type of notification provided.";
        }
        leaf message {
          type string;
          description
            "An optional human-readable value.";
        }
        container ssh-host-keys {
          description
            "A list of SSH host keys an NMS may use to authenticate
             a NETCONF connection to the device with.";
          list ssh-host-key {
            when "../type = bootstrap-complete" {
              description
                "SSH host keys are only sent when the notification
                 type is 'bootstrap-complete'.";
            }
            description
              "An SSH host-key";
            leaf format {
              type enumeration {
                enum ssh-dss { description "ssh-dss"; }
                enum ssh-rsa { description "ssh-rsa"; }
              }
              mandatory true;
              description
                "The format of the SSH host key.";
            }
            leaf key-data {
              type string;
              mandatory true;
              description
                "The key data for the SSH host key";
            }
          }
        }
        container trust-anchors {
```

```
            description
              "A list of trust anchor certificates an NMS may use to
               authenticate a NETCONF or RESTCONF connection to the
               device with.";
            list trust-anchor {
              when "../type = bootstrap-complete" {
                description
                  "Trust anchors are only sent when the notification
                   type is 'bootstrap-complete'.";
              }
              description
                "A list of trust anchor certificates an NMS may use to
                 authenticate a NETCONF or RESTCONF connection to the
                 device with.";
              leaf-list protocol {
                type enumeration {
                  enum netconf-ssh     { description "netconf-ssh"; }
                  enum netconf-tls     { description "netconf-tls"; }
                  enum restconf-tls    { description "restconf-tls"; }
                  enum netconf-ch-ssh  { description "netconf-ch-ssh"; }
                  enum netconf-ch-tls  { description "netconf-ch-tls"; }
                  enum restconf-ch-tls { description "restconf-ch-tls"; }
                }
                min-elements 1;
                description
                  "The protocols that this trust anchor secures.";
              }
              leaf certificate {
                type binary;
                mandatory true;
                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).";
              }
            }
          }
        }
      } // end action
```

```
    }
  }
}
```

<CODE ENDS>

8. Security Considerations

8.1. Immutable storage for trust anchors

Devices MUST ensure that all their trust anchor certificates, including those for the owner certificate and ownership voucher, 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.

8.2. Clock Sensitivity

The solution in this document relies on TLS certificates, owner certificates, ownership vouchers, and CRLs, all of which require an accurate clock in order to be processed correctly. Devices implementations should take care to ensure the devices have a reliable clock when processing signed data, ideally be using a built-in real time clock (RTC). If a device does not have an RTC, then it SHOULD try to use NTP to initialize its clock before processing any time-sensitive bootstrapping data. It is understood that NTP is itself unsecured, not enabling the client to authenticate the server, and therefore easily spoofed. In the case that NTP is spoofed, it is possible for a replay attack to occur where an ownership voucher assignment from a previous owner is replayed on a device that has since been claimed by a new owner. For this reason, for devices that do not contain an RTC, it is RECOMMENDED that manufacturers only issue a single ownership voucher for the lifetime of a device.

8.3. 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 (e.g., via a DNS service discovery lookup, where only a hostname or IP address is returned).

To compensate for this, this document requires that devices do not send their IDevID certificate for client authentication, and that they do not POST any progress notifications, and that they assert that data downloaded from the server is signed, just as bootstrapping data would need to be signed if read from a removable storage device.

8.4. Entropy loss over time

Section 7.2.7.2 of the IEEE Std 802.1AR-2009 standard says that IDevID certificate should never expire (i.e. having a notAfter 99991231235959Z). Given the long-lived nature of these certificates, it is paramount to use a strong key length (e.g., 512-bit ECC).

8.5. Serial Numbers

This draft suggests using the device's serial number as the unique identifier in its IDevID certificate. This is because serial numbers are ubiquitous and prominently contained in invoices and on labels affixed to devices and their packaging. That said, serial numbers many times encode revealing information, such as the device's model number, manufacture date, and/or sequence number. Knowledge of this information may provide an adversary with details needed to launch an attack.

9. IANA Considerations

9.1. The BOOTP Manufacturer Extensions and DHCP Options Registry

The following registrations are in accordance to RFC 2939 [RFC2939] for "BOOTP Manufacturer Extensions and DHCP Options" registry maintained at http://www.iana.org/assignments/bootp-dhcp-parameters.

9.1.1. DHCP v4 Option

Tag: XXX

Name: Zero Touch Redirect Information

Returns a YANG-defined redirect-information object, encoded in the encoding specified by 'encoding'. Currently only "xml" and "json" are supported.

Code	Len		
XXX	n	encoding	redirect-information

Reference: RFC XXXX

9.1.2. DHCP v6 Option

Tag: YYY

Name: Zero Touch Redirect Information

Returns a YANG-defined redirect-information object, encoded in the encoding specified by 'encoding'. Currently only "xml" and "json" are supported.

Code	Len		
XXX	n	encoding	redirect-information

Reference: RFC XXXX

9.2. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

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

9.3. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020]. Following the format defined in [RFC6020], the the following registration is requested:

name: ietf-zerotouch-bootstrap-server
namespace: urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server
prefix: ztbs
reference: RFC XXXX

10. Other Considerations

Both this document and [draft-pritikin-anima-bootstrapping-keyinfra] define bootstrapping mechanisms. The authors have collaborated on both solutions and believe that each solution has merit and, in fact, can work together. That is, it is possible for a device to support both solutions simultaneously.

11. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): David Harrington, Michael Behringer, Dean Bogdanovic, Martin Bjorklund, Joe Clarke, Toerless Eckert, Stephen Farrell, Stephen Hanna, Wes Hardaker, Russ Mundy, Reinaldo Penno, Randy Presuhn, Max Pritikin, Michael Richardson, Juergen Schoenwaelder.

Special thanks goes to Steve Hanna, Russ Mundy, and Wes Hardaker for brainstorming the original I-D's solution during the IETF 87 meeting in Berlin.

12. References

12.1. Normative References

[draft-ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home (work in progress)", draft-ieft-netconf-restconf-10 (work in progress), December 2015, <https://tools.ietf.org/html/draft-ietf-netconf-call-home-17>.

[draft-ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ieft-netconf-restconf-10 (work in progress), 2016, <https://tools.ietf.org/html/draft-ietf-netconf-restconf-10>.

[draft-ietf-netconf-server-model]
Watsen, K., "NETCONF Server Model (work in progress)", draft-ieft-netconf-server-model-09 (work in progress), March 2016, <http://tools.ietf.org/html/draft-ietf-netconf-call-home-17>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://www.rfc-editor.org/info/rfc1035>.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <http://www.rfc-editor.org/info/rfc1951>.

[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>.

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <http://www.rfc-editor.org/info/rfc2315>.

[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, <http://www.rfc-editor.org/info/rfc2782>.

[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>.

[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>.

[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, <http://www.rfc-editor.org/info/rfc6125>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <http://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <http://www.rfc-editor.org/info/rfc6763>.

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

[Std-802.1AR-2009] IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

12.2. Informative References

[draft-pritikin-anima-bootstrapping-keyinfra]
Pritikin, M., Behringer, M., and S. Bjarnason, "Bootstrapping Key Infrastructures", draft-pritikin-anima-bootstrapping-keyinfra-03 (work in progress), 2016, <https://tools.ietf.org/html/draft-pritikin-anima-bootstrapping-keyinfra>.

[RFC2939] Droms, R., "Procedures and IANA Guidelines for Definition of New DHCP Options and Message Types", BCP 43, RFC 2939, DOI 10.17487/RFC2939, September 2000, <http://www.rfc-editor.org/info/rfc2939>.

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

[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>.

[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, <http://www.rfc-editor.org/info/rfc6698>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

Appendix A. Examples

A.1. Ownership Voucher

Following describes an example data-model for an ownership voucher. Real vouchers are expected to be encoded in a manufacturer-specific format outside the of scope for this draft.

A tree diagram describing an ownership voucher:

```
module: ietf-zerotouch-ownership-voucher
   +--rw voucher
      +--rw owner-id      string
      +--rw unique-id*    string
      +--rw created-on    yang:date-and-time
      +--rw expires-on?   yang:date-and-time
      +--rw signature     string
```

The YANG module for this example voucher:

<CODE BEGINS> file "ietf-zerotouch-ownership-voucher@2016-04-06.yang"

```
module ietf-zerotouch-ownership-voucher {

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-zerotouch-ownership-voucher";
  prefix "ztov";

  import ietf-yang-types { 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 the format for a ZeroTouch ownership voucher,
    which is produced by Vendors, relayed by Bootstrap Servers, and
    consumed by devices.  The purpose of the voucher is to enable a
```

```
      device to ascertain the identity of its rightful owner, as
      certified by its Vendor.

      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 XXXX; see
      the RFC itself for full legal notices.";

    revision "2016-04-06" {
      description
       "Initial version";
      reference
       "RFC XXXX: Zero Touch Provisioning for NETCONF Call Home";
    }

    // top-level container
    container voucher {
      description
        "A voucher, containing the owner's identifier, a list of
         device's unique identifiers, information on when the
         voucher was created, when it might expire, and the
         vendor's signature over the above values.";
      leaf owner-id {
        type string;
        mandatory true;
        description
          "A Vendor-assigned value for the rightful owner of the
           devices enumerated by this voucher.  The owner-id value
           must match the value in the owner-certificate below";
      }
      leaf-list unique-id {
        type string;
        min-elements 1;
        description
          "The unique identifier (e.g., serial-number) for a device.
           The value must match the value in the device's IDevID
           certificate.  A device uses this value to determine if
           the voucher applies to it.";
      }
      leaf created-on {
```

```
        type yang:date-and-time;
        mandatory true;
        description
          "The date this voucher was created";
      }
      leaf expires-on {
        type yang:date-and-time;
        description
          "The date this voucher expires, if at all.  Use of this
           value requires that the device has access to a trusted
           real time clock";
      }
      leaf signature {
        type string;
        mandatory true;
        description
          "The signature over the concatenation of all the previous
           values";
      }
    }
  }

  <CODE ENDS>
```

Appendix B. Change Log

B.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 Zero Touch 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

B.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 Zero Touch Information DHCP Option
- Added IANA request for media types for boot-image and configuration

B.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

B.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.

B.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".

B.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

B.7. 05 to 06

- Minor update
- Added many Normative and Informative references.
- Added new section Other Considerations.

B.8. 06 to 07

- Minor update
- Added an Editorial Note section for RFC Editor.
- Updated the IANA Considerations section.

B.9. 07 to 08

- Minor update
- Updated to reflect review from Michael Richardson.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Mikael Abrahamsson
T-Systems

EMail: "mikael.abrahamsson@t-systems.se

NETCONF
Internet-Draft
Intended status: Informational
Expires: December 17, 2016

E. Voit
A. Clemm
A. Tripathy
E. Nilsen-Nygaard
A. Gonzalez Prieto
Cisco Systems
June 15, 2016

Restconf and HTTP Transport for Event Notifications
draft-voit-netconf-restconf-notif-00

Abstract

This document defines Event Notification YANG Subscription and Push mechanisms for Restconf, HTTP, and HTTP2 transports.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

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

Copyright Notice

Table of Contents

1. Introduction

Mechanisms to support Event subscription and push are defined in [rfc5277bis]. Enhancements to [rfc5277bis] which enable YANG Datastore subscription and push are defined in [yang-push]. This document provides a transport specification for these Restconf and HTTP. This has been driven by Requirements for subscriptions to YANG datastores are defined in [pub-sub-reqs].

Beyond based transport bindings, there are benefits which can be realized when transporting updates directly HTTP/2[RFC7540] which cn be realized via an implementation of this transport specification including:

- Subscription multiplexing over independent HTTP/2 streams
- Stream prioritization and stream dependencies
- Flow control on independent 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].

Configured Subscription: a Subscription installed via a configuration interface which persissts across reboots.

Data Node: An instance of management information in a datastore.

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

Dynamic Subscription: a Subscription negotiated between Subscriber and Publisher via create, establish, modify, and delete RPC control plane signaling messages.

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

Event Notification: a set of information intended for a Receiver indicating that one or more Event(s) have occurred. Details of the Event(s) may be included within.

Event Stream: a continuous, ordered set of Events grouped under an explicit criteria.

Notification: the communication of an occurrence, perhaps triggered by the occurrence of an Event.

Publisher: an entity responsible for streaming Event Notifications per the terms of a Subscription.

Receiver: a target to which a Publisher pushes Event Notifications. For Dynamic Subscriptions, the Receiver and Subscriber will often be the same entity.

Subscriber: an entity able to request and negotiate a contract for the receipt of Event Notifications 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.

3. Solution

Event subscription is defined in [rfc5277bis], YANG Datastore subscription is defined in [yang-push]. This section specifies transport mechanisms applicable to both.

3.1. Mechanisms for Subscription Establishment and Maintenance

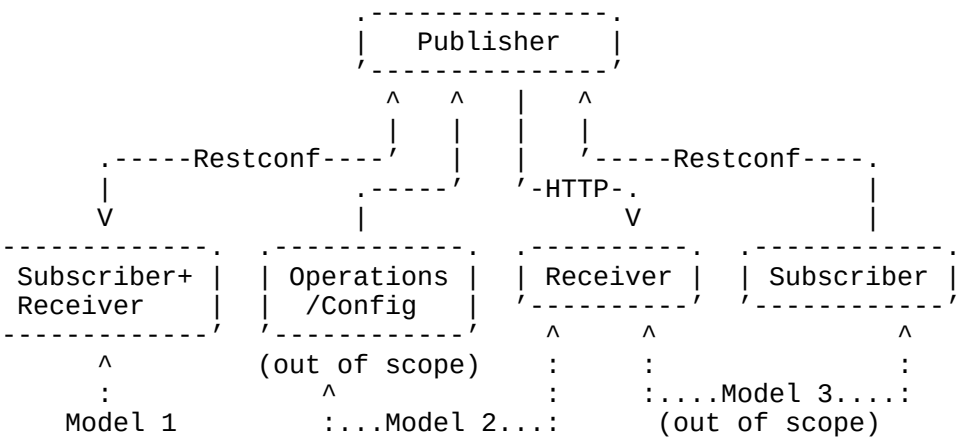
There are three models for Subscription establishment and maintenance:

1. Dynamic Subscription: Here the Subscriber and Receiver are the same. A Subscription ends with a subscription-terminated notification, or by a loss of transport connectivity.

2. Configured Subscription: Receiver(s) are specified on Publisher in startup and running config. Subscription is not terminated except via an operations interface. (Subscriptions may be Suspended, with no Event Notifications sent however.)

3. Proxy Subscription: Subscriber and Receiver are different. Subscription ends when a Subscription End-time is reached, or the Publisher process is restarted.

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.



3.1.1. Dynamic YANG Subscription over RESTCONF

Dynamic Subscriptions are configured and manage Subscriptions via signaling. This signaling is transported over [restconf]. Once established, streaming Event Notifications are then delivered via Restconf SSE.

3.1.2. Configured Subscription over HTTP

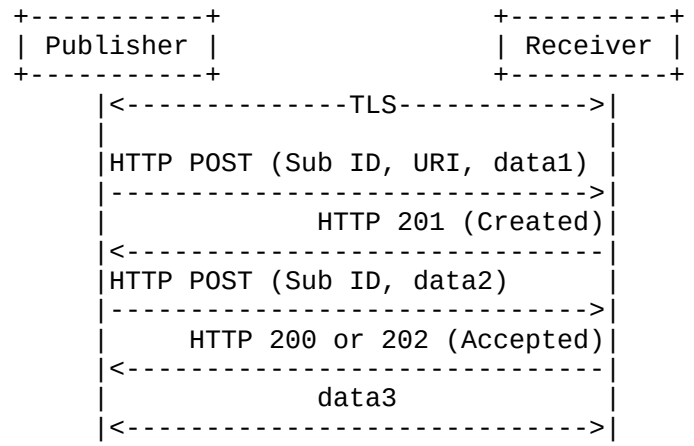
With a Configured Subscription, all information needed to establish a secure relationship with that Receiver is configured on the Publisher. With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing the Event Notifications to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that such Event Notifications be pushed via Restconf. Nor is there value which Restconf provides on top of HTTP. Therefore in place of Restconf, a TLS secured HTTP Client connection must be established with an HTTP Server located on the Receiver. Event Notifications will then be sent via HTTP Post messages to the Receiver.

Post messages will be addressed to HTTP augmentation code on the Receiver capable of accepting and responding to Event Notifications. At least the initial Post message must include the URI for the subscribed resource. This URI can be retained for operational tracking and debugging use by the Receiver.

After successful receipt of an initial Event Notification for a particular Subscription, the Reciever 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 2 depicts this message flow.



If HTTP/2 transport is available to a Receiver, the Publisher should also:

- point individual Event Notifications 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.

3.2. Subscription Multiplexing

When pushed directly over HTTP/2, it is expected that the Event Notifications from a single Subscription will be allocated a separate HTTP/2 stream. This 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, support may be appended on top of this specification.

With or without independent queueing of multiplexed subscriptions, it is possible that updates might be delivered in a different sequence than generated. Reasons for this might include (but are not limited to):

- replay of pushed updates
- temporary loss of transport connectivity, with update buffering and different dequeuing priorities per Subscription

- o population, marshalling and bundling of independent Subscription Updates, and

- o parallel HTTP1.1 sessions

Therefore each Event Notification 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 Event Notification 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 a Publisher.

3.3. Push Data Stream and Transport Mapping

Transported updates will contain data for one or more Event Notifications. Each transported Event Notification will contain several parameters:

- o A Subscription ID correlator

- o Event Notification(s) . (Note 1: These must be filtered per access control rules to contain only data that the Subscriber is authorized to see. Note 2: these Event Notifications might be Data Node Update(s).)

- o A timestamp indication when the Event Notification was generated on the Publisher.

3.3.1. Subscription and Updates via Restconf

Subscribers can dynamically learn whether a RESTCONF server supports various types of Event or Yang datastore subscription. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream. Some examples building upon the existing RESTCONF mechanisms are below:

```
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 any form of subscription, 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 may be selected from the Event Stream list provided in the capabilities exchange. Note that different encodings are supporting using different Event Stream locations. For example, the 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 uses the Server Sent Events[W3C-20121211] transport strategy to push filtered Event Notifications from the Event stream,.

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 pushed Event Notification 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
```

3.3.2. Subscription and Updates directly via HTTP

For any version of HTTP, the basic encoding will look as below. It consists of a JSON representation wrapped in an HTTP header.

```
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" }
    }
  }
}
```

4. 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 Event Notifications 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 Receivers notified.

A Subscription could be used to attempt retrieve information for which a Receiver has no authorized access. Therefore it is important that data pushed via a Subscription is authorized equivalently with regular data retrieval operations. Data being pushed to a Receiver needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model [RFC6536] applies even though the transport is not NETCONF.

One or more Publishers could be used to overwhelm a Receiver which doesn't even support Subscriptions. Therefore Event Notifications for Configured Subscriptions MUST only be transmittable over Encrypted transports. Clients which do not want pushed Event Notifications 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 Event Notifications 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.

5. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Andy Bierman, Sharon Chisholm, Yan Gang, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Hector Trevino, Kent Watsen, and Guangying Zheng.

6. References

6.1. Normative References

[restconf] Bierman, Andy., Bjorklund, Martin., and Kent. Watsen, "RESTCONF Protocol", March 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf/>.

[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", December 2015, <https://tools.ietf.org/html/draft-ietf-netconf-call-home-17>.

[pub-sub-reqs]
Voit, Eric., Clemm, Alexander., and Alberto. Gonzalez Prieto, "Subscribing to datastore push updates", February 2016, <https://datatracker.ietf.org/doc/draft-ietf-i2rs-pub-sub-requirements/>.

[rfc5277bis]
Gonzalez Prieto, Alberto., Clemm, Alexander., Voit, Eric., Prasad Tripathy, Ambika., and Einar. Nilsen-Nygaard, "NETCONF Event Notifications", March 2016, <https://datatracker.ietf.org/doc/draft-gonzalez-netconf-5277bis/>.

[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", March 2016, <https://datatracker.ietf.org/doc/draft-ietf-netmod-yang-json/>.

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

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

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

- An operator doesn’t want the subscription to be dependent on the maintenance of transport level keep-alives. (Transport independence provides different scalability characteristics.)

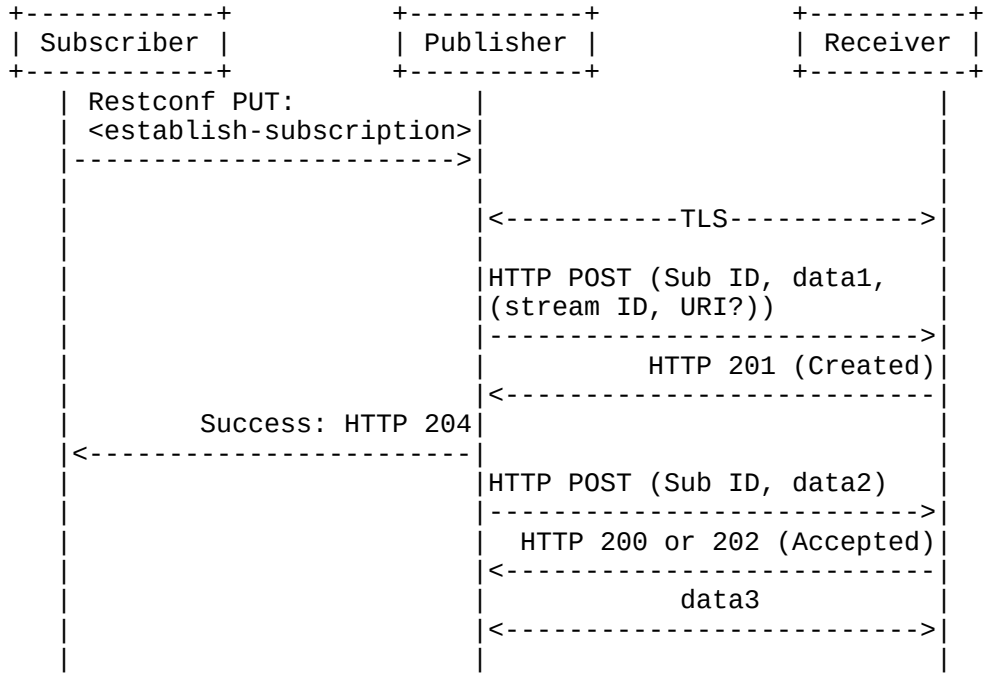
- o There is not a transport session binding, and a transient Subscription needs to survive in an environment where there is unreliable connectivity with the Receiver and/or Subscriber.

- o An operator wants the Publisher to include highly restrictive capacity management and Subscription security mechanisms outside of domain of existing operational or programmatic interfaces.

To build a Proxy Subscription, first the necessary information must be signaled as part of the <establish-subscription>. Using this set of Subscriber provided information; the same process described within section 3 will be followed. 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 a successful establishment, if the Subscriber wishes to track the state of Receiver subscriptions, it may choose to place a separate on-change Subscription into the "Subscriptions" subtree of the YANG Datastore on the Publisher.

Putting it all together, the message flow is:



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 when applicable.

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.

Appendix C. Issues being worked and resolved

(To be removed by RFC editor prior to publication)

C.1. Unresolved Issues

RT1 - Integration specifics for Restconf capability discovery on different types of Streams

RT2 - In what way to we position "Event notifications" model in this document vs. current solution in Restconf.

RT3 - Do we include 3rd party signaled subscriptions within models that need to be supported generically, or for a particular type of transport.

RT6 - We need to define encodings of rfc5277bis notifications for both Restconf and HTTP.

RT7 - HTTP native option doesn't currently use SSE. But we should evaluate moving to that as possible. It will make development integration easier and more consistent.

C.2. Agreement in principal

RT4 - Need to add into document examples of 5277bis Event streams. Document only includes yang-push examples at this point.

C.3. Resolved Issues

RT5 - Doesn't make sense to use Restconf for Configured subscriptions. HTTP will be used.

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