I2RS working group
Internet-Draft
Intended status: Informational
Expires: September 13, 2017

S. Hares
Huawei
A. Dass
Ericsson
March 12, 2017

NETCONF Changes to Support I2RS Protocol
draft-hares-netconf-i2rs-netconf-01.txt

Abstract

This document describes a NETCONF capabiilty to support the Interface to Routing system (I2RS) protocol requirements for I2RS protocol version 1. The I2RS protocol is a re-use higher layer protocol which defines extensions to other protocols (NETCONF and RESTCONf) and extensions to the Yang Data Modeling language.

The I2RS protocol supports ephemeral state datastores as control plane datastores. Initial versions of this document contain descriptions of the ephemeral datastore. Future versions may move this description to NETMOD datastore description documents.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 13, 2017.

Table of Contents

1. Introduction 3
2. Definitions Related to Ephemeral Configuration 3
2.1. Requirements language 4
2.2. I2RS Definitions 4
3. Requirements control-plane datstore and ephemeral capabilities 5
3.1. I2RS protocol requirements 5
3.2. Overview of NETCONF support for I2RS protocol requirements 5
4. NETCONF capability for control-plane datastore 5
4.1. Overview 6
4.2. Dependencies 7
4.3. Capability identifier 8
4.4. New Operations 8
4.4.1. <get-data> 8
4.4.2. <write data gt; 10
4.5. Modification to protocol operations 15
4.5.1. Unsupported protocol operations 15
4.5.2. Modified protocol operations 16
4.6. Interactions with Capabilities 16
4.6.1. Unsupported Capabiltiies 16
4.6.2. Modified Capabilities 16
5. NETCONF Ephemeral capability 17
5.1. Overview 17
5.2. Dependencies 18
5.3. New Operations 18
5.3.1. resource-limits 18
5.4. Modifications to Protocol Operations 18
5.4.1. Unsupported Operations 18
5.4.2. Modified Operations 19
6. Yang model Simple Ephemeral Data model 19
7. IANA Considerations 22
8. Security Considerations 22
9. Acknowledgements 22
10. References 22
10.1. Normative References: 22
10.2. Informative References 24
Authors' Addresses 27

1. Introduction

This a proposal for Yang additions to support the first version of the I2RS protocol.

The I2RS architecture [RFC7921] defines the I2RS interface "a programmatic interface for state transfer in and out of the Internet routing system". The I2RS protocol is a protocol designed to a higher level protocol comprised of a set of existing protocols which have been extended to work together to support a new interface to the routing system. The I2RS protocol is a "reuse" management protocol which creates new management protocols by reusing existing protocols and extending these protocols for new uses, and has been designed to be implemented in phases [RFC7921].

The first version of the I2RS protocol is comprised of extensions to existing features of NETCONF [RFC6241] and RESTCONF [I-D.ietf-netconf-restconf]. The data modeling language for the I2RS protocol will be Yang [RFC7950] with features and extensions proposed in this draft.

The structure of this document is:

Section 2 provides definitions for terms in this document.

Section 3 summarizes the I2RS requirements behind these changes.

Section 4 describes the NETCONF capability to support a control protocol datastore.

Section 5 the NETCONF capability to support ephemeral state. [I-D.ietf-i2rs-ephemeral-state] specifies the I2RS requirements for the ephemeral state.

Section 6 provides a Tiny Routing Rib Yang module used by the examples in this document.

2. Definitions Related to Ephemeral Configuration

This section reviews definitions from I2RS architecture [RFC7921] and NETCONF operational state definitions [I-D.ietf-netmod-revised-datastores] before using these to construct a definition of the ephemeral data store.

2.1. Requirements language

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].

2.2. I2RS Definitions

The I2RS architecture [RFC7921] defines the following terms:

ephemeral data: is data which does not persist across a reboot (software or hardware) or a power on/off condition. Ephemeral data can be configured data or data recorded from operations of the router. Ephemeral configuration data also has the property that a system cannot roll back to a previous ephemeral configuration state. (See [RFC7921] for an architectural overview, [I-D.ietf-i2rs-ephemeral-state] for requirements, and [I-D.ietf-netmod-revised-datastores] for discussion of how the ephemeral datastore as a control plane datastore interacts with intended datstore and dynamic configuration protocols to form the applied datastore".

local configuration: is the data on a routing system which does persist across a reboot (software or hardware) and a power on/off condition. Local configuration has the ability to roll back to a pervious configuration state. Local configuration is defined as the intended datastore [I-D.ietf-netmod-revised-datastores] which is modified by dynamic configuration protocols (such as DHCP) and the I2RS ephemeral data store

dynamic configuration protocols datastore are configuration protocols such as DHCP that interact with the intended datastore (which does persist across a reboot (software or hardware) power on/off condition), and the I2RS ephemeral state control plane datastore.

operator-applied policy: is a policy that an operator sets that determines how the ephemeral datastore as a control plane data store interacts with applied datastore (as defined in [I-D.ietf-netmod-revised-datastores]). This operator policy consists of setting a priority for each of the following (per [I-D.ietf-i2rs-ephemeral-state]):

* intended configuration,

* any dynamic configuration protocols,

* any control plane datastores (one of which is ephemeral.)

3. Requirements control-plane datstore and ephemeral capabilities

3.1. I2RS protocol requirements

The requirements for the I2RS protocol are defined in the following documents:

- I2RS Problem Statement [RFC7920],
- I2RS Architecture [RFC7921],
- I2RS Traceability [RFC7922],
- Publication and Subscription [RFC7923],
- I2RS Ephemeral State Requrements, , [I-D.ietf-i2rs-ephemeral-state]
- I2RS Protocol Security Requirements, [I-D.ietf-i2rs-protocol-security-requirements]

The Interface to the routing System (I2RS) creates a new capability for the routing systems, and with greater capaiblities come a greater need for security. The requirements for a secure environment for I2RS are described in [I-D.ietf-i2rs-security-environment-reqs].

3.2. Overview of NETCONF support for I2RS protocol requirements

This oveview reviews the following:

- Dependencies on Existing features
- Additions to use NETCONF [RFC6241] to support control plane datastores changes get to get data, write data,(via target [merge, replace, create, delete, copy, delete-all]), close-session, kill-session, rollback-on-error (all-or-nothing), validate (validation + roll-back-on-error (all-or-nothing),
- Additions for I2RS ephemeral
- NETCONF [RFC6241] changes to obtain control-plane datastore.

4. NETCONF capability for control-plane datastore

capability-name: control-plane

4.1. Overview

This capability defines the NETCONF protocol extensions for use with control plane protocols.

A control plane datastore is not part of the configuration datastore per [I-D.ietf-netmod-revised-datastores]. The control plane datastore many contain configuration and operational state. A router implementation may merge the configuration from a control plane datastore with configuration data from the configuration datastore. A query of the applied datastore will provide a list of the installed configuration from all datastores with meta data. The current architectural provides an origin identityref with the following mappimg to datastores for the

- static - configuration data store.
- dynamic - dynamic configuration or dynamic control plane datastores.
- system - created by system,
- data-model - created by "default" in use.

Clearly, the dynamic origin title is not enough to uniquely identify a control plane datastore entry in the applied datastore. Additional definitions will need to be added to the architectural model, but this will be specified in another document.

The control plane datastores do not restrict multiple access via the locking mechanisms (<lock> and <unlock>), but use a priority scheme to handle multiple clients attempting to write the same data. The default validation within a control plane datastore’s config objects (e.g. config=TRUE) is the configuration datastore validation, but if Yang data modules specify different validation for the datastore or specific nodes then the control plane datastores will use this validation.

Some data modules may be used for both a control plane datastore and the configuration datastore. If additional validation is used for these modules, it is recommend that these modules use the "rpc" function for the additional validation rather than the <write-data> functions.

4.2. Dependencies

The following are the dependencies for the :control-plane capability

- Yang features:
  - [I-D.ietf-netmod-revised-datastores] functionality including the ietf-yang-architecture" data module.
  - [I-D.hares-netmod-i2rs-yang] Yang additions related to datastores definitions related to control plane datastores (datastoredef, datastore, dstype, precedence, protosup, validation), and ephemeral state.
- The following NETCONF features:
  - NETCONF [RFC6241] with its updates [RFC7803],
  - Network Access Control Model [RFC6536] with update by [I-D.ietf-netconf-rfc6536bis]
  - Running NETCONF over TLS with mutually X.509 authentication [RFC7589]
  - Keystore Model [I-D.ietf-netconf-keystore],
  - Subscribing to Yang Datastore updates [I-D.ietf-netconf-yang-push],
  - NETCONF support for Event Notifications [I-D.ietf-netconf-netconf-event-notifications],
  - Subscribing to NETCONF Events (updated) [I-D.ietf-netconf-rfc5277bis]
  - Yang Patch Media type [I-D.ietf-netconf-yang-patch],
  - NETCONF/RESTCONF Zero Touch provisioning [I-D.ietf-netconf-zerotouch],
  - TLS Client and Server Models [I-D.ietf-netconf-tls-client-server]
  - Call Home [I-D.ietf-netconf-call-home],
  - Module library [RFC7895],

* NETCONF/RESTCONF Zero Touch provisioning [I-D.ietf-netconf-zerotouch],

4.3. Capability identifier

The controlplane-datastore capability is identified by the following capability string: (:control-plane (uri-tbd)) where the uri-tbd is to be assigned by IANA.

4.4. New Operations

The following are additional protocol operations NETCONF [RFC6241] to support the following queries based on a datastore source/target datastore being specified:

- "get-data"
- "write-data"
- "validate-data"

The <target-datastore> must be registered with IANA.

4.4.1. <get-data>

The get-data command has obtains configuration and operational data. The parameters the following:

source name of the datastore being queried. The valid names are "applied", "opstate", or a datstore name registered with IANA.

filter this identifies the portions of the device configuration datastore is to receive. If this parameter is not present, the entire datastore is returned. The filter MAY support subtypes "subtree", "uri", and "xpath" capabilities described in [RFC6241]. Filters may also include the elements for state (E.g. config true, config false, ephemeral true; ephemeral false;).

Positive Response If the device was able to satify the request, an <rpc-reply> is sent. The <data> section contains the appropriate subset.

Negative Response If the device was unable to satify the request, an <rpc-error> is included in the <rpc-reply>

Example - retrieve route1 in route list.
wfrom control plane datastore (cp-alpha)
and gets both configuration and ephemeral data.

```
 <rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data/
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                  <route-list>
                     <route-index>1</route-index>
                  </route-list<
      </filter>
    </get-data>
 </rpc>

  <rpc-rply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data<
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                  <route-list>
                     <route-index>1</route-index>
                         <prefix-match>192.2/8 /16<prefix-match>
                         <nexthop>129.1.5.1</nexthop>
                         <if-outgoing>Eth0</if-outgoing>
                         <installed>true</installed>
                  </route-list<
      </filter>
    </get-data>
 </rpc>
```

Figure 1

Example 2 - retrieve users subtree from the ephemeral database which has example control plane datastore (cp-alpha) and gets only config=true data;

```
 <rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data/
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                <route-list>
                    <route-index>1</route-index>
                 </route-list<
                type="state" select "config";
      </filter>
    </get-data>
 </rpc>

  <rpc-rply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-data<
      <source>
        <cp-alpha/>
      </source>
      <filter type="subtree">
      <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                 <route-list>
                    <route-index>1</route-index>
                        <prefix-match>192.2/8 /16<prefix-match>
                        <nexthop>129.1.5.1</nexthop>
                        <if-outgoing>Eth0</if-outgoing>
                 </route-list<
      </filter>
    </get-data>
 </rpc>
```

Figure 2 - get config data

4.4.2. <write data gt;

The write data operationals has the attributes of operation parameters, and parameters of target datastore, default-operations, test-option, error-option, a priority, secondary-id, config, and

opstate. The attributes of the operation for individual nodes wutg "config-true" are: create, delete, merge, remove, and replace. The attributes for all-datastore operations are create-datastore, copy-datastore, delete-datastore. The operations handle multiple-client writes by using priority values rather than a locking mechanism. If two or more clients interact over changing the data node, the priority values arbitrate between the the clients where the greatest priority (1=lowest) wins. The following operations can enact changes in opstate data nodes: create, delete, remove, reset.

The default-operations parameter flags are: merge, replace, or none. The test-option parameters flags are: test-then-set, set, and test-only. The error-option oarameter flags are: stop-on-error, continue-on-error, and rollback-on-error. The priority parameter is a integer giving the priority (range 1..5000).

4.4.2.1. Limitations based on other capability flags

The test-option parameters MAY only be set if the device advertises the :validate:1.1 capability. If test-option is set without the :validate:1.1 capability, an error is returned "no support for test-option".

The error-option subparameter "rollback-on-error" is enabled only if the :rollback-on-error capability is set and the data is under the config parameter.

A URL is accepted within the <source> or <target> parameters if the :url capability is set. An XPATH is accepted within the <source> or <target> parameters if the :xpath capability is set.

4.4.2.2. defaults

The following are the defaults:

- The target datastore does not exists, it will be created if local support exists. Otherwise, the reply will indicate "non-supported datastore".

- If no sub-operations is specified the sub-operation, the default is merge.

- If no priority parameter is specified, the priority will be set to 1 (lowest external priority).

- If error-option is not set, then the default is "stop-on-error". (Note: for I2RS WG input. Is "stop-on-error" the same as "none"?)

- If no test-option parameter is set, the write data operates as a ’set" without validation first.

- if no secondary-identifier is given, the secondary identifier stored is "" indicating a null string.

The NETCONF priority for the "write data" function is simply the Netconf priority range. If implementations have an internal priority, the precedence between the local configuration and the NETCONF supplied configuration is set by a operator applied knob. For example, an implementation could indicate that the local configuration priority can range from 0-10 and the NETCONF priority is 10 plus the value of the priority parameter.

4.4.2.3. target

The target is a datastore whose name is registered with IANA.

Example Write data

dsname = i2rs-agent

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <write data>
    <target><i2rs-agent></target>
        <operation>merge</operation>
    <priority>50<</priority>
        <error-option>all-or-nothing</error-option>
        <config>
             <top xmlns:t=
         "http://example.com/schema/1.0/i2rs/tiny-rt-instance">
                  <route-list>
                     <route-index>1</route-index>
                         <prefix-match>192.2/8 /16<prefix-match>
                         <nexthop>129.1.5.1</nexthop>
                         <if-outgoing>Eth0</if-outgoing>
                  </route-list<
              </top>
        </config>
   </write data>
```

4.4.2.4. write data operations attributes

The description of each operation is below:

<create> (config, opstate) : the creation of the data node in the datatstore if and only if the data does not already exist in the target datastore. If the data already exists, then an <rpc-error> is return wtih an error tag of "data-exists". Failure information or Success informatnoi is also pass to the notification functions (events and traceability).

<create-datastore > (config, opstate) : the creation of the datastore based on datastructures in the config and opstate parameters. The datastore ownership is set to client creating the datastore plus the priority.

<delete> (config) : the deletion of the data node if the data node exists in the configuration and either the same client deletes the node or a client with a high priority deletes the node. If configuration data does not exist in the datastore, then the <rpc-error> element is returned with a <error-tag> with value of "data-missing". The error information is passed to the notification functions to be sent as event or (optionally) placed in a tracing file.

<delete-datastore> Delete all configuration and operational data configured into the datastore, and the delete the datastore. The client requesting a delete-store must either be the owner of the datastore or have a higher priority than the client that owns the datastore. If a higher priority client takes ownership, the lower priority client is notified. If the devices is able to satisfy request, the positive response is <rcp-reply> that includes <ok> element. If the device is unable to complete request, the <rcp-reply> that includes <rpc-error> element. The operations results are forwarded to event and traceabilty functions.

<copy-datastore> If the datastore does not exist, it creates the datastore and copies the configuration values and opstate values into the datastore. The ownership information (client identity and priority) is saved as part of the datastore. If the datastore does exist and the client with ownership of the datastore changes it, then the client can replace all the datastore nodes. If a different client with lower priority than the client having ownership wants to change the datastore, the request is rejected. If a client with higher priority than the client having ownership, then the the owership changed to the new client, all the data in the datastore is deleted, all new data uploaded (config and opstate nodes). If a server is able to satisfy request, the

positive response is <rcp-reply> that includes <ok> element. If the server is unable to complete request, the <rcp-reply> that includes <rpc-error> element. The operational results are forwarded to event and traceabilty functions. If a copy-datastore action is in progress, and a client with a higher priority asks to copy-datastore, the original

<merge> (config) : parameter specifies merge. If the <priority< specifies The current data is modified by the new data in a merge of the data based on priorities. If the same client merges the data, priority is ignored. If a different client merges the data, the priority must be created than the current client's priority. If any data is replaced, this event is passed to the notification and traceability functions to pass to the setting client and the client that set the original value.

<remove> (config, opstate) : the remove of the data node if the data nodes specified in the <config> or the <opstaste> node exist. If data nodes do not exist, the "remove" operation is silently ignored and error results are forwarded to traceabilty functions.

<replace> (config) : replaces data in target if the same client replaces the top-level node, priority is ignored. If a different client replaces the note, the priority must be higher than the top level node's priority. If any data is replaced, this event is passed to the notification function (events and traceability), and a notification is sent to the previous client setting this data that the data has been reset. If the request to replace is reject due to the current top-level node having a higher priority, then an <rpc-error> returns with an error tag of "insufficient-priority". If the node is replace by a different client, the original client is notified of the change.

<reset> (opstate) : resets opstate nodes with counters to initial settings.

4.4.2.5. <priority parameters>

The priority parameter sets a integer value for the priority as shown in figure x.

4.4.2.6. <test-op parameter>

The <test-option> parameter performs basic function it does in the <edit-config> basic function. Just in [RFC6241], the <test-option> parameter MAY be specified only if the device advertises the ":validate:1.1" capability. The only difference is that the validation specified by the data model may augment the validation

test and the valdiation will also include the ability of the client to set this element. If a validation error occurs, the test-then-set will not perform the write-data function.

4.4.2.7. <error-option> parameter

<error-option> has the following attributes

stop-on-error : Abort the write-data on the first error. This is the default.

msg-rollback-on-error : if an error condition occurs such that error serverity <rpc-error> is generated, the server will stop processing the write-data operation and restore the specific configuratoin to its complete state at the start of the "write-data" operation. This option only processes roll-back on single messages which includes

* if multiple operations occur in single message, error in one operation (E.g. read data) must not impact other operations (write-data);

* multiple operations in multiple message should be supported, but roll-back should only include a single message.

This option requires the server to support the :rollback-on-error capability.

4.4.2.8. secondary-id

This operation associates a secondary identifier with a set of write-data operations. The secondary identifier is an opaque string.

4.5. Modification to protocol operations

4.5.1. Unsupported protocol operations

The following protocol operations are not supported in the control plane datastore:

o <get-config>,

o <edit-config>,

o <copy-config>,

o <delete-config>,

- o <lock>,
- o <unlock>

4.5.2. Modified protocol operations

4.5.2.1. <close-session> and <kill-session>

The <close-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

The <kill-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

4.6. Interactions with Capabilities

4.6.1. Unsupported Capabiltiies

The following capabilities are not supported:

writeable-running capability,

candidate configuration capability,

confirmed commit capability,

distinct startup capbility,

4.6.2. Modified Capabilities

4.6.2.1. rollback-on-error

The rollback-on-error allows the error handling to be roll-back-on-error (all-or-nothing in I2RS terms) for the control plane datastore. The control plane datastore name is a valid target if the rollback-on-error capability is combined with the control plane datastore capability.

4.6.2.2. validate

The validation capability engated with the control plane capability operates to validate the config portion of the control plane datastore. Therefore, the <target> is allowed to have a datstore name which is registered with IANA.

The validation of the configuration portion may contain the "validation" yang command which provides alternative validation mechanisms for specific data objects.

4.6.2.3. URL capability and XPATH capability

The URL capabilities specify a <url> in the <source> and <target> operate as normal, but are allowed to specify a module within a control plane datastore.

5. NETCONF Ephemeral capability

capability-name: control-plane

5.1. Overview

The ephemeral capability is the ability to support control plane datastores which are entirely ephemeral or have ephemeral state modules, or ephemeral statements within objects in a modules. These objects can be configuration state (config=TRUE) or operational state (config=FALSE).

Ephemeral state in datastores, ephemeral modules or ephemeral objects within a module have one key characteristics: the data does not persist across reboots. The ephemeral configuration state must be restored by a client, and the operational state will need to be regenerated.

The entire requirements for ephemeral state for the I2RS control plane protocol are listed in [I-D.ietf-i2rs-ephemeral-state]. Many of these require fulfilled by the NETCONF control-plane capabilty(Ephemeral-REQ-07, Ephemeral-REQ-11, Ephemeral-REQ-12, Ephemeral-REQ-13, Ephemeral-REQ-14, Ephemeral-REQ-16).

The key features include:

- references between (to/from) ephemeral state and non-ephemeral state for constraints purposes (see Ephemeral-REQ-02, Ephemeral-REQ-03, and Ephemeral-REQ-04 in [I-D.ietf-i2rs-ephemeral-state]).

- operations to set and modify the constraints on the amount of resources the I2RS Agent (aka NETCONF server) can consume (Ephemeral-REQ-05)

- Yang modules must identify Yang objects (modules, submodules or objects within yang modules which are ephemeral and augment other nodes) and allow an "ephemeral=TRUE" feature.

5.2. Dependencies

Ephemeral state is not supported in the configuration datstore. The ephemeral state capability depends on having the control-plane datastore capability enabled (with appropriate NETCONF capabilities described above), and an IANA registered datastore name.

Yang must support the ability to to denote that a datastore, module, submodule or object within a module can be denoted as ephemeral. This capbility depends on the yang additions described in [I-D.hares-netmod-i2rs-yang] for control plane datastores, ephemeral key word, and validation key word.

Ephemeral state operation depends on notification of events and traceability of errors. I2RS ephemeral state requires that

5.3. New Operations

Note: One operation that is suggested for ephemeral state is to set resource limits. It does not seem to be an ephemeral state issue, but a control plane issue. This feature is placed here until future discussion for I2RS WG.

5.3.1. resource-limits

resource-limits

definition - TBD

The [I-D.ietf-i2rs-ephemeral-state] suggests setting these limits, but it does not seem to be an ephemeral function.

5.4. Modifications to Protocol Operations

5.4.1. Unsupported Operations

The ephemeral state only works as an augment to the control-plane datastore. Therefore, the following protocol operations, which are not supported in the control-plane datastore capability, are also not supported in the ephemeral capability:

- <get-config>,
- <edit-config>,
- <copy-config>,
- <delete-config>,

- o <lock>,
- o <unlock>

5.4.2. Modified Operations

The ephemeral state only works as an augment to the control-plane datastore with specific ephemeral validations. Therefore, the <close-session> and <kill-session> are modified as described in the sections below.

5.4.2.1. <close-session> Modifications

The <close-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

5.4.2.2. <close-session> Modifications

The <kill-session> is modified to take a target of a control plane datastore name (registered with IANA). Since no locks are set, none should be released.

5.4.2.3. <validate> Modifications

The ephemeral state may require validation to determine if the constraints obey ephemeral-state rules. If the :validate capability is used, the following parameter requires ephemeral-state contraints (Ephemeral-REQ-02, Ephemeral-REQ-03, and Ephemeral-REQ-04). If the ephemeral-constraint parameter is engaged for a module or object that is not ephemeral, the parameter is silently ignored. Error information is forwarded to the event notification processes and the traceability functions.

Additional Parameter

ephemeral-constraint

6. Yang model Simple Ephemeral Data model

```
datastoredef cp-alpha {
   dstype control-plane;
   description {
          "example control plane datastore ";
     }
   module-list tiny-rt-instance;
   precedence applied {
     precedenceval 5;
```

```
      }
      precedence opstate {
         precedenceval 5;
      }
   }

   datastoredef cp-beta {
      dstype control-plane i2rs-v0;
      description {
             "example control plane datastore ";
        }
      module-list tiny-rib;
      ephemeral true;
      precedence applied {
        precedenceval 50;
      }
      precedence opstate {
         precedenceval 50;
      }
   }
   module cp-example-1 {
    namespace "http://exaple.com/schema/cp-examples/1.1/tiny-rib";
    prefix trib;
    import ietf-inet-types {
       prefix inet;
    }
    import ietf-yang-types {
       prefix yang;
     }

    grouping trib-rt {
       description "tiny rib route";
       leaf route-index  {
              type uint64;
              mandatory true;
              description "route index";
            }
           leaf v4-prefix-match {
                   type inet:ipv4-prefix;
                   mandatory true;
           }
           leaf v4-nexthop {
               type inet:ipv4
                   mandatory true;
              }
           leaf if-outgoing {
               type if:interface-ref;
                   mandatory true;
```

```
                description {
                 "Name of outgoing interface";
                }
          leaf installed {
                type boolean;
          config false;
                description "rt install status ";
             }
          }
       container tiny-rt-instance {
         description
                "Tiny routing instance for
                     example purposes";
           leaf name {
              type string;
                  description
                   "The name of routing instance
            which must be unique. ";
         }
           list route-list {
             key "route-index";
             description
               "a list of routes of rib"
             uses trib-rt;
           }
       }

       rpc trib-add {
        description "add route to tiny rib";
           input {
             leaf datastore {
            type string; //iana registered
            mandatory true;
            description
                      "iana datastore name";
             }
             container trib-routes {
                description
                      "Tiny rib routes to be added
                       to tiny rib";
                 list route-list {
                   key "route-index";
                   users trib-rt;
                 }
             }
         }
           output {
             container trib-add-status {
```

```
                leaf success {
                     type boolean;
                     description "add succeded";
                    }
                    leaf failure-reason {
                     type string;
                     description "reason for failure ";
                    }
               }
          }
     }
```

7. IANA Considerations

TBD

8. Security Considerations

The security requirements for the I2RS protocol are covered in [I-D.ietf-i2rs-protocol-security-requirements]. The security environment the I2RS protocol is covered in [I-D.ietf-i2rs-security-environment-reqs]. Any person implementing or deploying the I2RS protocol should consider both security requirements.

9. Acknowledgements

TBD

10. References

10.1. 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>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC5339] Le Roux, JL., Ed. and D. Papadimitriou, Ed., "Evaluation of Existing GMPLS Protocols against Multi-Layer and Multi-Region Networks (MLN/MRN)", RFC 5339, DOI 10.17487/RFC5339, September 2008, <http://www.rfc-editor.org/info/rfc5339>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[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>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

[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>.

[RFC7158] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7158, DOI 10.17487/RFC7158, March 2014, <http://www.rfc-editor.org/info/rfc7158>.

[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>.

[RFC7803] Leiba, B., "Changing the Registration Policy for the NETCONF Capability URNs Registry", BCP 203, RFC 7803, DOI 10.17487/RFC7803, February 2016, <http://www.rfc-editor.org/info/rfc7803>.

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

[RFC7920] Atlas, A., Ed., Nadeau, T., Ed., and D. Ward, "Problem Statement for the Interface to the Routing System", RFC 7920, DOI 10.17487/RFC7920, June 2016, <http://www.rfc-editor.org/info/rfc7920>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <http://www.rfc-editor.org/info/rfc7921>.

[RFC7922] Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", RFC 7922, DOI 10.17487/RFC7922, June 2016, <http://www.rfc-editor.org/info/rfc7922>.

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

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <http://www.rfc-editor.org/info/rfc7952>.

[RFC7958] Abley, J., Schlyter, J., Bailey, G., and P. Hoffman, "DNSSEC Trust Anchor Publication for the Root Zone", RFC 7958, DOI 10.17487/RFC7958, August 2016, <http://www.rfc-editor.org/info/rfc7958>.

10.2. Informative References

[I-D.hares-netmod-i2rs-yang]
Hares, S. and a. amit.dass@ericsson.com, "Yang for I2RS Protocol", draft-hares-netmod-i2rs-yang-04 (work in progress), March 2017.

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-23 (work in progress), November 2016.

[I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-17 (work in progress), September 2016.

[I-D.ietf-i2rs-rib-data-model]
Wang, L., Ananthakrishnan, H., Chen, M., amit.dass@ericsson.com, a., Kini, S., and N. Bahadur, "A YANG Data Model for Routing Information Base (RIB)", draft-ietf-i2rs-rib-data-model-07 (work in progress), January 2017.

[I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-10 (work in progress), December 2016.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-03 (work in progress), March 2017.

[I-D.ietf-i2rs-yang-l3-topology]
Clemm, A., Medved, J., Varga, R., Liu, X., Ananthakrishnan, H., and N. Bahadur, "A YANG Data Model for Layer 3 Topologies", draft-ietf-i2rs-yang-l3-topology-08 (work in progress), January 2017.

[I-D.ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), December 2015.

[I-D.ietf-netconf-keystore]
Watsen, K. and G. Wu, "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), October 2016.

[I-D.ietf-netconf-netconf-event-notifications]
Prieto, A., Clemm, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF Support for Event Notifications", draft-ietf-netconf-netconf-event-notifications-01 (work in progress), October 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[I-D.ietf-netconf-rfc5277bis]
Clemm, A., Prieto, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "Subscribing to Event Notifications", draft-ietf-netconf-rfc5277bis-01 (work in progress), October 2016.

[I-D.ietf-netconf-rfc6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-00 (work in progress), January 2017.

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-01 (work in progress), November 2016.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-14 (work in progress), November 2016.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-05 (work in progress), March 2017.

[I-D.ietf-netconf-zerotouch]
Watsen, K. and M. Abrahamsson, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-12 (work in progress), January 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", draft-ietf-netmod-revised-datastores-00 (work in progress), December 2016.

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-04 (work in progress), March 2017.

[I-D.ietf-netmod-syslog-model]
Wildes, C. and K. Koushik, "A YANG Data Model for Syslog Configuration", draft-ietf-netmod-syslog-model-12 (work in progress), February 2017.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com

I2RS working group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

S. Hares
Huawei
A. Dass
Ericsson
March 13, 2017

RESTCONF Changes to Support I2RS Protocol
draft-hares-netconf-i2rs-restconf-01.txt

Abstract

This document describes two RESTCONF optional capabilities (control plane datastore capability, ephemeral state capabilities) that are needed to support the I2RS protocol needs. The I2RS protocol requirese an ephemeral control plane datastore. as control plane datastores.

The purpose of this draft is to kick-start the discussions with NETCONF on these two capabilities.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Table of Contents

1. Introduction 2
1.1. Background on I2RS 3
1.2. Structure of draft 3
2. Definitions and Background on I2RS 3
2.1. IETF Requirements language 3
2.2. I2RS Definitions 3
2.3. Example for operator-applied policy 5
2.4. I2RS protocol requirements 6
3. RESTCONF control plane datastore capability 6
3.1. Overview 6
3.2. Dependencies 7
3.3. New Operations 7
3.4. Modified Operations 7
4. RESTCONF protocol extensions for the ephemeral datastore . . 8
4.1. Overview 8
4.2. Dependencies 9
4.3. Capability identifier 9
4.4. New Operations 9
4.5. modification to data resources 9
4.6. Modification to existing operations 10
5. IANA Considerations 10
6. Security Considerations 10
7. Acknowledgements 11
8. References 11
8.1. Normative References: 11
8.2. Informative References 13
Authors' Addresses 16

1. Introduction

This a proposal for the following two RESTCONF capabilities to augment RESTCONF [RFC8040] to support the first version of the I2RS protocol: Control plane datstore capability and ephemeral state capability. The yang that supports this proposal is described in [I-D.hares-netmod-i2rs-yang]. This work is based on the datastore definitions in [I-D.ietf-netmod-revised-datastores].

This draft parallels a similar proposal for NETCONF [RFC6241] is described in [I-D.hares-netconf-i2rs-protocol]. The difference is the original design is to embedded the I2RS multi-headed collision resolution in the "control plane data store capability". However

RESTCONF has edit-collision capability already which only needs a usage description. Therefore, this document has a I2RS Edit-Collision capability.

Caveat: This work is an individual draft (not an I2RS WG effort)

1.1. Background on I2RS

The I2RS architecture [RFC7921] defines the I2RS interface "a programmatic interface for state transfer in and out of the Internet routing system". The I2RS protocol is a protocol designed to a higher level protocol comprised of a set of existing protocols which have been extended to work together to support a new interface to the routing system. The I2RS protocol is a "reuse" management protocol which creates new management protocols by reusing existing protocols and extending these protocols for new uses, and has been designed to be implemented in phases [RFC7921].

1.2. Structure of draft

The structure of this document is:

Section 2 provides definitions and background on I2RS work. (If you are familiar with the I2RS architecture and requirements, you can skip this section.)

Section 3 describes the RESTCONF control plane datastore capability.

Section 4 describes the RESTCONF ephemeral state capabiilty. .

2. Definitions and Background on I2RS

This section reviews definitions from I2RS architecture, and provides background on I2RS work for the reader.

2.1. IETF Requirements language

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].

2.2. I2RS Definitions

The I2RS architecture [RFC7921] defines the following terms:

ephemeral data: is data which does not persist across a reboot (software or hardware) or a power on/off condition. Ephemeral

data can be configured data or data recorded from operations of the router. Ephemeral configuration data also has the property that a system cannot roll back to a previous ephemeral configuration state. (See [RFC7921] for an architectural overview, [I-D.ietf-i2rs-ephemeral-state] for requirements, and [I-D.ietf-netmod-revised-datastores] for discussion of how the ephemeral datastore as a control plane datastore interacts with intended datstore and dynamic configuration protocols to form the applied datastore".

local configuration: is the data on a routing system which does persist across a reboot (software or hardware) and a power on/off condition. Local configuration has the ability to roll back to a pervious configuration state. Local configuration is defined as the intended datastore [I-D.ietf-netmod-revised-datastores] which is modified by dynamic configuration protocols (such as DHCP) and the I2RS ephemeral data store

dynamic configuration protocols datastore are configuration protocols such as DHCP that interact with the intended datastore (which does persist across a reboot (software or hardware) power on/off condition), and the I2RS ephemeral state control plane datastore.

control plane protocols datastore is a datastore which is loaded by control plane protocols (e.g. I2RS protocol) rather than system configuration protocols. (see [I-D.ietf-netmod-revised-datastores]).

operator-applied policy: is a policy that an operator sets that determines how the ephemeral datastore as a control plane data store interacts with applied datastore (as defined in [I-D.ietf-netmod-revised-datastores]). This operator policy consists of policy knobs that the operator sets to determine how the I2RS agent control plane ephemeral state datastore will interact with the intended configuration datastor and the dynamic configuration protocol datastore. Three policy knobs could be used to implement this policy:

* policy knob 1: I2RS Ephemeral control-plane datastore takes takes precedence over the intended datastore in the routing protocols.

* policy knob 2: Updated intended configuration datastore takes precedence over the I2RS ephemeral control-plane data store in the routing protocols

* policy knob 3: Ephemeral control plane datastore takes precedence over any other dynamic configuration protocols datastore.

2.3. Example for operator-applied policy

An practical example for three states of the operator-applied policy may help the reader understand the concept. Consider the following three desired outcomes with their policy knob states:

Monitoring Features only The policy knob settings are:

Policy knob 1=false,

policy knob 2=true,

Policy knob 3=false,

Action: I2RS protocol software feature is installed, but the operator does not want the I2RS ephemarl datastore to take precedence (that is be used) on any variables in the applied configuration datastore. This policy set might be valid if I2RS is only suppose to monitor data on this node through newly defined parameters.

I2RS Agent Changes win the policy knob settings would be:

Policy knob 1=true,

policy knob 2=false,

Policy knob 3=false,

Action: This is the normal case for the I2RS Agent where the ephemeral control-plane datastore takes precedence over the intended configuration datastore and dynamic configuration datastores. The values from the I2RS ephemearl datastore are used rather than the intended configuration datastore and the dynamic configuration protocol datastore. When the ephemeral data is removed by the I2RS agent, the dyanmic configuration datastore and the intended configuration datastore state is restored, combined and passed to the routing protocols for application.

Just change until next configuration update the policy knob settings would be:

Policy knob 1=true,

policy knob 2=true,

Policy knob 3=false,

Action: This case can occur if the I2RS Client write to the ephemeral control plane data store is only suppose to take precedence until the next configuration cycle from a centralized system. Suppose the local configuration is get by the centralized system at 11:00pm each night. The I2RS Client writes temporary changes to the routing system via the I2RS agent ephemeral write. At 11:00pm, the local configuration update overwrite the ephemeral. The I2RS Agent notifies the I2RS Client which is tracking which of the ephemeral changes are being overwritten.

2.4. I2RS protocol requirements

The requirements for the I2RS protocol are defined in the following documents:

- I2RS Problem Statement [RFC7920],
- I2RS Architecture [RFC7921],
- I2RS Traceability [RFC7922],
- Publication and Subscription [RFC7923],
- I2RS Ephemeral State Requrements, , [I-D.ietf-i2rs-ephemeral-state]
- I2RS Protocol Security Requirements, [I-D.ietf-i2rs-protocol-security-requirements]

The Interface to the routing System (I2RS) creates a new capability for the routing systems, and with greater capaiblities come a greater need for security. The requirements for a secure environment for I2RS is described in [I-D.ietf-i2rs-security-environment-reqs].

3. RESTCONF control plane datastore capability

capability-name: control-plane datastore

3.1. Overview

The :control-plane datastore capability enables the RESTCONF to support the following:

- API resource that is {+restconf/cp-data} - the storage of control plane datastore's configuration that includes configuration ({+restconf/cp-data/config}) and operational state specific to the control plane datastore ({+restconf/cp-data/opstate}).

- It also includes the ability to have the applied datastore and the opstate datatstore (per [I-D.ietf-netmod-revised-datastores].

3.2. Dependencies

This protocol strawman utilizes the following existing proposed features for NETCONF and RESTCONF

- RESTCONF [RFC8040].

- Module library [RFC7895],

- RESTCONF Patch Media Type [RFC8072],

- NETCONF Support for event notifications [I-D.ietf-netconf-netconf-event-notifications],

- Publication/Subscription via Push [I-D.ietf-netconf-yang-push],

- NETCONF and HTTP Transport for Event Notivications [I-D.ietf-netconf-restconf-notif],

- Publication/Subscription via Push [I-D.ietf-netconf-yang-push],

- syslog yang module (both [RFC5424] and [I-D.ietf-netmod-syslog-model]

3.3. New Operations

none

3.4. Modified Operations

All RESTCONF methods (OPTIONS, HEAD, GET, POST, PT, PATCH, DELETE) need to work in the control plane datastores. config=TRUE data, and where appropriate config=FALSE data.

Editor's Note: Amazingly, RESTCONF is almost ready for the control plane data. The authors understanding of the I2RS protocol needs is why. The best situation is to ask the RESTCONF authors for help specifying what needs to change for the RESTCONF to allow references to datastore.

4. RESTCONF protocol extensions for the ephemeral datastore

capability-name: ephemeral-state

4.1. Overview

This capability defines the RESTCONF protocol extensions for control plane protocols that support control plane data stores with ephemeral data.

Ephemeral state is not unique to I2RS work.

The ephemeral capability is the ability to support control plane datastores which are entirely ephemeral or have ephemeral state modules, or ephemeral statements within objects in a modules. These objects can be configuration state (config=TRUE) or operational state (config=FALSE).

Ephemeral state in datastores, ephemeral modules or ephemeral objects within a module have one key characteristics: the data does not persist across reboots. The ephemeral configuration state must be restored by a client, and the operational state will need to be regenerated.

The entire requirements for ephemeral state for the I2RS control plane protocol are listed in [I-D.ietf-i2rs-ephemeral-state]. Compared to RESTCONF functionality there are 4 groups of additional changes:

Constraints The ability to enforce the constraints for references (to/from) the {+restconf/data} datastore, and a {+restconf/cp-data} control plane datastore. ((see Ephemeral-REQ-02, Ephemeral-REQ-03, and Ephemeral-REQ-04 in [I-D.ietf-i2rs-ephemeral-state]). The "validation" yang statement in [I-D.hares-netmod-i2rs-yang] could encode specific validation for the ephemeral case per datatstore or per object. [Editor’s note: Aid is needed from NETCONF authors to determine the best way to enforce the constraints.]

Library Tracking of Epheemral Yang modules must identify Yang objects (modules, submodules or objects within yang modules which are ephemeral and augment other nodes) and allow an "ephemeral=TRUE" feature.

Roll-back an ephemeral node cannot roll-back to its previous value,

4.2. Dependencies

The ephemeral capabilities have the following dependencies:

- Yang modules must support the following:
  - identifying datastores, modules, and objects as ephemeral. (ephemeral=True)
  - Albility to have control plane datastores which are ephemeral.
- The following features must be supported by RESTCONF
  - Module library [RFC7895],
  - RESTCONF Protocol [RFC8040],
  - RESTCONF Patch Media Type [RFC8072],
  - NETCONF Support for event notifications [I-D.ietf-netconf-netconf-event-notifications],
  - Publication/Subscription via Push [I-D.ietf-netconf-yang-push],
  - NETCONF and HTTP Transport for Event Notivications [I-D.ietf-netconf-restconf-notif],
  - Subsribing to Yang datastore push updates [I-D.ietf-netconf-yang-push],

4.3. Capability identifier

The ephemeral-datastore capability is identified by the following capability string: ephemeral (TBD URI)

4.4. New Operations

none

4.5. modification to data resources

RESTCONF must be able to support the ephemeral data in a control plane datastore

RESTCONF library functions must be able to store an indication that a data module has ephemeral state.

4.6. Modification to existing operations

The current operations in RESTCONF are: OPTIONS, HEAD, GET, POST, PUT, PATCH, and DELETE.

Editor's note: From here is not solutions but a list of features to discuss with the RESTCONF team.

The operations must support the following things about ephemeral The ephemeral data store has the following general qualities:

1. The ephemeral datastore is never locked. (RESTCONF does not use a locking mechanism.)

2. The ephemeral portion of the intended configuration, applied state, and derived state does not persist over a reboot,

3. an ephemeral node cannot roll-back to its previous value,

4. Since ephemeral data store is just data that does not presist over a reboot, then in theory any node or group of nodes in a YANG data model could be ephemeral. The YANG data module must indicate what portion of the data model (if any) is ephemeral.

    * A YANG data module could be all ephemeral (e.g. [I-D.ietf-i2rs-rib-data-model]) with no directly associated configuration models,

    * A YANG model could be all ephemeral but associated with a configuration model

    * or a single data node or data tree could be made ephemeral.

5. The management protocol (NETCONF/RESTCONF) needs to signal which poritons of a data model(node, tree, or data model) are ephemeral in the module library [RFC7895].

5. IANA Considerations

This is a protocol strawman - nothing is going to IANA.

6. Security Considerations

The security requirements for the I2RS protocol are covered in [I-D.ietf-i2rs-protocol-security-requirements]. The security environment the I2RS protocol is covered in [I-D.ietf-i2rs-security-environment-reqs]. Any person implementing

or deploying the I2RS protocol should consider both security requirements.

7. Acknowledgements

TBD

8. References

8.1. 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>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <http://www.rfc-editor.org/info/rfc4107>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC5339] Le Roux, JL., Ed. and D. Papadimitriou, Ed., "Evaluation of Existing GMPLS Protocols against Multi-Layer and Multi-Region Networks (MLN/MRN)", RFC 5339, DOI 10.17487/RFC5339, September 2008, <http://www.rfc-editor.org/info/rfc5339>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, DOI 10.17487/RFC5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[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>.

[RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <http://www.rfc-editor.org/info/rfc6244>.

[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>.

[RFC7158] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7158, DOI 10.17487/RFC7158, March 2014, <http://www.rfc-editor.org/info/rfc7158>.

[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>.

[RFC7803] Leiba, B., "Changing the Registration Policy for the NETCONF Capability URNs Registry", BCP 203, RFC 7803, DOI 10.17487/RFC7803, February 2016, <http://www.rfc-editor.org/info/rfc7803>.

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

[RFC7920] Atlas, A., Ed., Nadeau, T., Ed., and D. Ward, "Problem Statement for the Interface to the Routing System", RFC 7920, DOI 10.17487/RFC7920, June 2016, <http://www.rfc-editor.org/info/rfc7920>.

[RFC7921] Atlas, A., Halpern, J., Hares, S., Ward, D., and T. Nadeau, "An Architecture for the Interface to the Routing System", RFC 7921, DOI 10.17487/RFC7921, June 2016, <http://www.rfc-editor.org/info/rfc7921>.

[RFC7922] Clarke, J., Salgueiro, G., and C. Pignataro, "Interface to the Routing System (I2RS) Traceability: Framework and Information Model", RFC 7922, DOI 10.17487/RFC7922, June 2016, <http://www.rfc-editor.org/info/rfc7922>.

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

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <http://www.rfc-editor.org/info/rfc7952>.

[RFC7958] Abley, J., Schlyter, J., Bailey, G., and P. Hoffman, "DNSSEC Trust Anchor Publication for the Root Zone", RFC 7958, DOI 10.17487/RFC7958, August 2016, <http://www.rfc-editor.org/info/rfc7958>.

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

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

8.2. Informative References

[I-D.hares-netconf-i2rs-protocol]
Hares, S. and a. amit.dass@ericsson.com, "NETCONF Changes to Support I2RS Protocol", draft-hares-netconf-i2rs-protocol-00 (work in progress), November 2016.

[I-D.hares-netmod-i2rs-yang]
Hares, S. and a. amit.dass@ericsson.com, "Yang for I2RS Protocol", draft-hares-netmod-i2rs-yang-04 (work in progress), March 2017.

[I-D.ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", draft-ietf-i2rs-ephemeral-state-23 (work in progress), November 2016.

[I-D.ietf-i2rs-protocol-security-requirements]
Hares, S., Migault, D., and J. Halpern, "I2RS Security Related Requirements", draft-ietf-i2rs-protocol-security-requirements-17 (work in progress), September 2016.

[I-D.ietf-i2rs-rib-data-model]
Wang, L., Ananthakrishnan, H., Chen, M., amit.dass@ericsson.com, a., Kini, S., and N. Bahadur, "A YANG Data Model for Routing Information Base (RIB)", draft-ietf-i2rs-rib-data-model-07 (work in progress), January 2017.

[I-D.ietf-i2rs-rib-info-model]
Bahadur, N., Kini, S., and J. Medved, "Routing Information Base Info Model", draft-ietf-i2rs-rib-info-model-10 (work in progress), December 2016.

[I-D.ietf-i2rs-security-environment-reqs]
Migault, D., Halpern, J., and S. Hares, "I2RS Environment Security Requirements", draft-ietf-i2rs-security-environment-reqs-04 (work in progress), March 2017.

[I-D.ietf-i2rs-yang-l3-topology]
Clemm, A., Medved, J., Varga, R., Liu, X., Ananthakrishnan, H., and N. Bahadur, "A YANG Data Model for Layer 3 Topologies", draft-ietf-i2rs-yang-l3-topology-08 (work in progress), January 2017.

[I-D.ietf-netconf-call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", draft-ietf-netconf-call-home-17 (work in progress), December 2015.

[I-D.ietf-netconf-keystore]
Watsen, K. and G. Wu, "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), October 2016.

[I-D.ietf-netconf-netconf-event-notifications]
Prieto, A., Clemm, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF Support for Event Notifications", draft-ietf-netconf-netconf-event-notifications-01 (work in progress), October 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[I-D.ietf-netconf-restconf-notif]
Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Clemm, A., and A. Bierman, "Restconf and HTTP Transport for Event Notifications", draft-ietf-netconf-restconf-notif-02 (work in progress), March 2017.

[I-D.ietf-netconf-rfc5277bis]
Clemm, A., Prieto, A., Voit, E., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "Subscribing to Event Notifications", draft-ietf-netconf-rfc5277bis-01 (work in progress), October 2016.

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-01 (work in progress), November 2016.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-14 (work in progress), November 2016.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-05 (work in progress), March 2017.

[I-D.ietf-netconf-zerotouch]
Watsen, K. and M. Abrahamsson, "Zero Touch Provisioning for NETCONF or RESTCONF based Management", draft-ietf-netconf-zerotouch-13 (work in progress), March 2017.

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-01 (work in progress), March 2017.

[I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-04 (work in progress), March 2017.

[I-D.ietf-netmod-syslog-model]
Wildes, C. and K. Koushik, "A YANG Data Model for Syslog Configuration", draft-ietf-netmod-syslog-model-13 (work in progress), March 2017.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Amit Daas
Ericsson

Email: amit.dass@ericsson.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

K. Watsen
Juniper Networks
March 13, 2017

Keystore Model
draft-ietf-netconf-keystore-01

Abstract

This document defines a YANG data module for a system-level keystore 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.

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

- "VVVV" --> 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:

- "2017-03-13" --> 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

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on September 14, 2017.

Table of Contents

1. Introduction 3
1.1. Requirements Language 3
1.2. Tree Diagram Notation 3
2. The Keystore Model 4
2.1. Overview 4
2.2. Example Usage 5
2.3. YANG Module 10
3. Design Considerations 20
4. Security Considerations 21
5. IANA Considerations 22
5.1. The IETF XML Registry 22
5.2. The YANG Module Names Registry 22
6. Acknowledgements 23
7. References 23
7.1. Normative References 23
7.2. Informative References 23
Appendix A. Change Log 25
A.1. server-model-09 to 00 25
A.2. keychain-00 to keystore-00 25
A.3. 00 to 01 25
Appendix B. Open Issues 25

Author's Address 25

1. Introduction

This document defines a YANG [RFC6020] data module for a system-level keystore 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 keystore utility to implement this module.

1.1. Requirements Language

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 Diagram Notation

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 Keystore Model

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

```
  module: ietf-keystore
      +--rw keystore
         +--rw keys
         |  +--rw key* [name]
         |     +--rw name                                    string
         |     +--rw algorithm-identifier                    identityref
         |     +--rw private-key                             union
         |     +--ro public-key                              binary
         |     +--rw certificates
         |     |  +--rw certificate* [name]
         |     |     +--rw name     string
         |     |     +--rw value?   binary
         |     +---x generate-certificate-signing-request
         |        +---w input
         |        |  +---w subject       binary
         |        |  +---w attributes?   binary
         |        +--ro output
         |           +--ro certificate-signing-request    binary
         +--rw trusted-certificates* [name]
         |  +--rw name                   string
         |  +--rw description?           string
         |  +--rw trusted-certificate* [name]
         |     +--rw name           string
         |     +--rw certificate?   binary
         +--rw trusted-host-keys* [name]
            +--rw name                string
            +--rw description?        string
            +--rw trusted-host-key* [name]
               +--rw name        string
               +--rw host-key    binary

    notifications:
      +---n certificate-expiration
         +--ro certificate        instance-identifier
         +--ro expiration-date    yang:date-and-time
```

2.2. Example Usage

The following example illustrates what a fully configured keystore 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.

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

```
     <!-- private keys and associated certificates -->
     <keys>
       <key>
         <name>ex-rsa-key</name>
         <algorithm-identifier>rsa1024</algorithm-identifier>
         <private-key>Base64-encoded RSA Private Key</private-key>
         <public-key>Base64-encoded RSA Public Key</public-key>
         <certificates>
           <certificate>
             <name>ex-rsa-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
         </certificates>
       </key>

       <key>
         <name>tls-ec-key</name>
         <algorithm-identifier>secp256r1</algorithm-identifier>
         <private-key>Base64-encoded EC Private Key</private-key>
         <public-key>Base64-encoded EC Public Key</public-key>
         <certificates>
           <certificate>
             <name>tls-ec-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
         </certificates>
       </key>

       <key>
         <name>tpm-protected-key</name>
         <algorithm-identifier>rsa2048</algorithm-identifier>
         <private-key>Base64-encoded RSA Private Key</private-key>
         <public-key>Base64-encoded RSA Public Key</public-key>
         <certificates>
           <certificate>
             <name>builtin-idevid-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
           <certificate>
             <name>my-ldevid-cert</name>
             <value>Base64-encoded PKCS#7</value>
           </certificate>
         </certificates>
       </key>
     </keys>

     <!-- trusted netconf/restconf client certificates -->
     <trusted-certificates>
```

```
      <name>explicitly-trusted-client-certs</name>
      <description>
        Specific client authentication certificates for explicitly
        trusted clients.  These are needed for client certificates
        that are not signed by a trusted CA.
      </description>
      <trusted-certificate>
        <name>George Jetson</name>
        <certificate>Base64-encoded X.509v3</certificate>
      </trusted-certificate>
    </trusted-certificates>

    <trusted-certificates>
      <name>explicitly-trusted-server-certs</name>
      <description>
        Specific server authentication certificates for explicitly
        trusted servers.  These are needed for server certificates
        that are not signed by a trusted CA.
      </description>
      <trusted-certificate>
        <name>Fred Flintstone</name>
        <certificate>Base64-encoded X.509v3</certificate>
      </trusted-certificate>
    </trusted-certificates>

    <!-- trust anchors (CA certs) for authenticating clients -->
    <trusted-certificates>
      <name>deployment-specific-ca-certs</name>
      <description>
        Trust anchors (i.e. CA certs) that are used to authenticate
        client connections.  Clients are authenticated if their
        certificate has a chain of trust to one of these configured
        CA certificates.
      </description>
      <trusted-certificate>
        <name>ca.example.com</name>
        <certificate>Base64-encoded X.509v3</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>Base64-encoded X.509v3</certificate>
      </trusted-certificate>
    </trusted-certificates>

    <!-- trusted SSH host keys -->
    <trusted-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>Base64-encoded OneAsymmetricKey</host-key>
      </trusted-host-key>
    </trusted-host-keys>

  </keystore>
```

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">
      <keystore
        xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
        <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>
        </keystore>
      </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-keystore">
         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 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-keystore">
       <certificate>/ks:keystore/ks:private-keys/ks:private-key\
       /ks:certificate-chains/ks:certificate-chain/ks: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-keystore@2017-03-13.yang"

module ietf-keystore {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-keystore";
  prefix "ks";

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

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

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

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

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


  description
   "This module defines a keystore 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 "2017-03-13" {
    description
     "Initial version";
    reference
     "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
                Models";
  }

  // Identities

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

  identity rsa1024 {
    base key-algorithm;
    description
      "The RSA algorithm using a 1024-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa2048 {
    base key-algorithm;
    description
```

```
      "The RSA algorithm using a 2048-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa3072 {
    base key-algorithm;
    description
      "The RSA algorithm using a 3072-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa4096 {
    base key-algorithm;
    description
      "The RSA algorithm using a 4096-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa7680 {
    base key-algorithm;
    description
      "The RSA algorithm using a 7680-bit key.";
    reference
      "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
                RSA Cryptography Specifications Version 2.1.";
  }

  identity rsa15360 {
    base key-algorithm;
    description
      "The RSA algorithm using a 15360-bit key.";
    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.";
  }

  // data model

  container keystore {
    nacm:default-deny-write;
    description
      "The keystore contains both active material (e.g., private keys
       and passwords) and passive material (e.g., trust anchors).

       The active material can be used to support either a server (e.g.,
       a TLS/SSH server's private) or a client (a private key used for
       TLS/SSH client-certificate based authentication, or a password
       used for SSH/HTTP-client authentication).

       The passive material can be used to support either a server
       (e.g., client certificates to trust) or clients (e.g., server
       certificates to trust).";

    container keys {
```

```
      description
        "A list of keys maintained by the keystore.";
      list key {
        key name;
        description
          "A key maintained by the keystore.";
        leaf name {
          type string;
          description
            "An arbitrary name for the key.";
        }
        leaf algorithm-identifier {
          type identityref {
            base "key-algorithm";
          }
          mandatory true;
          description
            "Identifies which algorithm is to be used with the key.
             This value determines how the 'private-key' and 'public-
             key' fields are interpreted.";
             // no params, such as in RFC 5912? (no are set for algs
             // we care about, but what about the future?
        }
        leaf private-key {
          nacm:default-deny-all;
          type union {
            type binary;
            type enumeration {
              enum "RESTRICTED" {
                description
                 "The private key is restricted due to access-control.";
              }
              enum "INACCESSIBLE" {
                description
                 "The private key is inaccessible due to being protected
                  by the cryptographic hardware modules (e.g., a TPM).";
              }
            }
          }
          mandatory true;
          description
            "A binary string that contains the value of the private
             key. The interpretation of the content is defined in the
             registration of the key algorithm.  For example, a DSA key
             is an INTEGER, an RSA key is represented as RSAPrivateKey
             as defined in [RFC3447], and an Elliptic Curve Cryptography
             (ECC) key is represented as ECPrivateKey as defined in
             [RFC5915]"; // text lifted from RFC5958
```

```
        }

        // no key usage (ref: RFC 5912, pg 101 -- too X.509 specific?)

        leaf public-key {
          type binary;
          config false;
          mandatory true;
          description
            "A binary string that contains the value of the public
             key. The interpretation of the content is defined in the
             registration of the key algorithm.  For example, a DSA key
             is an INTEGER, an RSA key is represented as RSAPublicKey
             as defined in [RFC3447], and an Elliptic Curve Cryptography
             (ECC) key is represented using the 'publicKey' described in
             [RFC5915]";
        }
        container certificates {
          description
            "Certificates associated with this private key.  More
             than one certificate per key is enabled to support,
             for instance, a TPM-protected key that has associated
             both IDevID and LDevID certificates.";
          list certificate {
            key name;
            description
              "A certificate for this private key.";
            leaf name {
              type string;
              description
                "An arbitrary name for the certificate.  The name
                 must be a unique across all keys, not just within
                 this key.";
            }
            leaf value {
              type binary;
              description
               "An unsigned PKCS #7 SignedData structure, as specified
                by Section 9.1 in RFC 2315, containing just certificates
                (no content, signatures, or CRLs), encoded using ASN.1
                distinguished encoding rules (DER), as specified in
                ITU-T X.690.

                This structure contains, in order, the certificate
                itself and all intermediate certificates leading up
                to a trust anchor certificate.  The certificate MAY
                optionally include the trust anchor certificate.";
              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 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).";

            }
          }
        }
      }
    }

    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-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  // is this the correct type?
            "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).";
```

```
        }
      }
    }
  }

  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.

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

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. This being the case, the top-level node in this module is marked with the NACM value 'default-deny-write'.

/keystore/keys/key/private-key: When writing this node, implementations MUST ensure that the strength of the key being configured is not greater than the strength of the underlying secure transport connection over which it is communicated. Implementations SHOULD fail the write-request if ever the strength of the private key is greater then the strength of the underlying transport, and alert the client that the strength of the key may have been compromised. Additionally, when deleting this node, 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.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or

notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

/keystore/keys/key/private-key: This node is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The best reason for returning this node is to support backup/restore type workflows. This being the case, this node is marked with the NACM value 'default-deny-all'.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

generate-certificate-signing-request: For this RPC operation, it is RECOMMENDED that implementations 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.

5. IANA Considerations

5.1. 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-keystore
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-keystore
namespace: urn:ietf:params:xml:ns:yang:ietf-keystore
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

[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>.

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

7.2. Informative References

[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>.

[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>.

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

[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.

A.2. keychain-00 to keystore-00

- Renamed module from "keychain" to "keystore" (Issue #3)

A.3. 00 to 01

- Replaced the 'certificate-chain' structures with PKCS#7 structures. (Issue #1)
- Added 'private-key' as a configurable data node, and removed the 'generate-private-key' and 'load-private-key' actions. (Issue #2)
- Moved 'user-auth-credentials' to the ietf-ssh-client module. (Issues #4 and #5)

Appendix B. Open Issues

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

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Networks
J. Schoenwaelder
Jacobs University Bremen
March 13, 2017

NETCONF Client and Server Models
draft-ietf-netconf-netconf-client-server-02

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:

- I-D.ietf-netconf-keystore
- I-D.ietf-netconf-ssh-client-server
- I-D.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 I-D.ietf-netconf-ssh-client-server
- "ZZZZ" --> the assigned RFC value for I-D.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:

- "2017-03-13" --> 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

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Terminology 4
1.2. Tree Diagrams 4
2. The NETCONF Client Model 4
2.1. Tree Diagram 5
2.2. Example Usage 8
2.3. YANG Model 10
3. The NETCONF Server Model 19
3.1. Tree Diagram 20
3.2. Example Usage 23
3.3. YANG Model 26
4. Design Considerations 37
4.1. Support all NETCONF transports 37
4.2. Enable each transport to select which keys to use 37
4.3. Support authenticating NETCONF clients certificates . . . 37
4.4. Support mapping authenticated NETCONF client certificates to usernames 38
4.5. Support both listening for connections and call home . . 38
4.6. For Call Home connections 38
4.6.1. Support more than one NETCONF client 38
4.6.2. Support NETCONF clients having more than one endpoint 38
4.6.3. Support a reconnection strategy 38
4.6.4. Support both persistent and periodic connections . . 39
4.6.5. Reconnection strategy for periodic connections . . . 39
4.6.6. Keep-alives for persistent connections 39
4.6.7. Customizations for periodic connections 39
5. Security Considerations 39
6. IANA Considerations 41
6.1. The IETF XML Registry 41
6.2. The YANG Module Names Registry 41
7. Acknowledgements 41
8. References 42
8.1. Normative References 42
8.2. Informative References 43
Appendix A. Change Log 44
A.1. server-model-09 to 00 44
A.2. 00 to 01 44
A.3. 01 to 02 44
Appendix B. Open Issues 44
Authors' Addresses 44

1. Introduction

This document defines two YANG [RFC7950] 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 [RFC8071]).

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 [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

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 endpoints
        |     |  |     |  +--rw endpoint* [name]
        |     |  |     |     +--rw name       string
        |     |  |     |     +--rw address    inet:host
        |     |  |     |     +--rw port?      inet:port-number
        |     |  |     +--rw server-auth
        |     |  |     |  +--rw trusted-ssh-host-keys?
        |     |  |     |  |       -> /ks:keystore/trusted-host-keys/name
        |     |  |     |  +--rw trusted-ca-certs?        leafref
        |     |  |     |  |       {sshcom:ssh-x509-certs}?
        |     |  |     |  +--rw trusted-server-certs?    leafref
        |     |  |     |          {sshcom:ssh-x509-certs}?
        |     |  |     +--rw client-auth
        |     |  |     |  +--rw username?      string
        |     |  |     |  +--rw (auth-type)?
        |     |  |     |     +--:(certificate)
        |     |  |     |     |  +--rw certificate?   leafref
        |     |  |     |     |          {sshcom:ssh-x509-certs}?
        |     |  |     |     +--:(public-key)
        |     |  |     |     |  +--rw public-key?
        |     |  |     |     |          -> /ks:keystore/keys/key/name
        |     |  |     |     +--:(password)
        |     |  |     |        +--rw password?      union
        |     |  |     +--rw transport-params
        |     |  |             {ssh-client-transport-params-config}?
        |     |  |        +--rw host-key
        |     |  |        |  +--rw host-key-alg*   identityref
        |     |  |        +--rw key-exchange
        |     |  |        |  +--rw key-exchange-alg*   identityref
        |     |  |        +--rw encryption
```

```
|     |  |        |  +--rw encryption-alg*   identityref
|     |  |        +--rw mac
|     |  |        |  +--rw mac-alg*   identityref
|     |  |        +--rw compression
|     |  |           +--rw compression-alg*   identityref
|     |  +--:(tls) {tls-initiate}?
|     |     +--rw tls
|     |        +--rw endpoints
|     |        |  +--rw endpoint* [name]
|     |        |     +--rw name       string
|     |        |     +--rw address    inet:host
|     |        |     +--rw port?      inet:port-number
|     |        +--rw server-auth
|     |        |  +--rw trusted-ca-certs?       leafref
|     |        |  +--rw trusted-server-certs?   leafref
|     |        +--rw client-auth
|     |        |  +--rw (auth-type)?
|     |        |     +--:(certificate)
|     |        |        +--rw certificate?   leafref
|     |        +--rw hello-params
|     |                 {tls-client-hello-params-config}?
|     |           +--rw tls-versions
|     |           |  +--rw tls-version*   identityref
|     |           +--rw cipher-suites
|     |              +--rw cipher-suite*   identityref
|     +--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
+--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?
|     |  |       -> /ks:keystore/trusted-host-keys/name
|     |  +--rw trusted-ca-certs?        leafref
|     |  |       {sshcom:ssh-x509-certs}?
|     |  +--rw trusted-server-certs?    leafref
|     |          {sshcom:ssh-x509-certs}?
|     +--rw client-auth
|     |  +--rw username?      string
|     |  +--rw (auth-type)?
|     |     +--:(certificate)
|     |     |  +--rw certificate?   leafref
|     |     |          {sshcom:ssh-x509-certs}?
|     |     +--:(public-key)
|     |     |  +--rw public-key?
|     |     |          -> /ks:keystore/keys/key/name
|     |     +--:(password)
|     |        +--rw password?      union
|     +--rw transport-params
|             {ssh-client-transport-params-config}?
|        +--rw host-key
|        |  +--rw host-key-alg*   identityref
|        +--rw key-exchange
|        |  +--rw key-exchange-alg*   identityref
|        +--rw encryption
|        |  +--rw encryption-alg*   identityref
|        +--rw mac
|        |  +--rw mac-alg*   identityref
|        +--rw compression
|           +--rw compression-alg*   identityref
|  +--:(tls) {tls-listen}?
|     +--rw tls
|        +--rw address?        inet:ip-address
|        +--rw port?           inet:port-number
|        +--rw server-auth
|        |  +--rw trusted-ca-certs?       leafref
|        |  +--rw trusted-server-certs?   leafref
|        +--rw client-auth
|        |  +--rw (auth-type)?
|        |     +--:(certificate)
|        |        +--rw certificate?   leafref
|        +--rw hello-params
|                {tls-client-hello-params-config}?
|           +--rw tls-versions
|           |  +--rw tls-version*   identityref
|           +--rw cipher-suites
```

```
                              +--rw cipher-suite*   identityref
```

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 [I-D.ietf-netconf-keystore].

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

  <!-- NETCONF servers to initiate connections to -->
  <initiate>
    <netconf-server>
      <name>corp-fw1</name>
      <ssh>
        <endpoints>
          <endpoint>
            <name>corp-fw1.example.com</name>
            <address>corp-fw1.example.com</address>
          </endpoint>
          <endpoint>
            <name>corp-fw2.example.com</name>
            <address>corp-fw2.example.com</address>
          </endpoint>
        </endpoints>
        <server-auth>
          <trusted-server-certs>deployment-specific-ca-certs</trusted-server-cer
ts>
        </server-auth>
        <client-auth>
          <username>foobar</username>
          <public-key>ex-rsa-key</public-key>
        </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-h
ost-keys>
        </server-auth>
        <client-auth>
          <username>foobar</username>
          <public-key>ex-rsa-key</public-key>
        </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@2017-03-13.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-ssh-client {
    prefix ss;
    revision-date 2017-03-13; // stable grouping definitions
    reference
      "RFC YYYY: SSH Client and Server Models";
  }

  import ietf-tls-client {
    prefix ts;
    revision-date 2017-03-13; // 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>

    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 clients.

    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 "2017-03-13" {
    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 8071: 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 8071: 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 initiating 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.";
              uses endpoints-container {
                refine endpoints/endpoint/port {
                  default 830;
                }
              }
              uses ss:ssh-client-grouping;
            }
          } // end ssh

          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:tls-client-grouping;
            }
          } // end tls

        } // end transport

        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
                  server. If the connection goes down, immediately
                  start trying to reconnect to it, using the
                  reconnection strategy.
```

```
        This connection type minimizes any NETCONF server
        to NETCONF client 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 client 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 server.  An
           unresponsive SSH/TLS server will be dropped after
           approximately max-attempts * max-wait seconds.";
        reference
          "RFC 8071: 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
            server, a SSH/TLS-level message will be sent
            to test the aliveness of the SSH/TLS server.";
        }
        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 server before assuming the SSH/TLS
            server is no longer alive.";
        }
      }
```

```
            }
          }
          case periodic-connection {
            container periodic {
              presence true;
              description
               "Periodically connect to the NETCONF server, so that
                the NETCONF server may deliver messages pending for
                the NETCONF client.  The NETCONF server must close
                the connection when it is ready to release it. Once
                the connection has been closed, the NETCONF client
                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 client will wait before re-establishing
                  a connection to the NETCONF server.  The NETCONF
                  client may initiate a connection before this
                  time if desired (e.g., to set configuration).";
              }
            }
          }
        }
      }
      container reconnect-strategy {
        description
         "The reconnection strategy directs how a NETCONF client
          reconnects to a NETCONF server, after discovering its
          connection to the server has dropped, even if due to a
```

```
            reboot.  The NETCONF client 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
                   clients SHOULD be able to remember the last
                   endpoint connected to across reboots.";
              }
            }
            default first-listed;
            description
             "Specifies which of the NETCONF server's endpoints the
              NETCONF client should start with when trying to connect
              to the NETCONF server.";
          }
          leaf max-attempts {
            type uint8 {
              range "1..max";
            }
            default 3;
            description
             "Specifies the number times the NETCONF client tries to
              connect to a specific endpoint before moving on to the
              next endpoint in the list (round robin).";
          }
        }
      } // end netconf-server
    } // 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.";
        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.";
              leaf address {
                type inet:ip-address;
                description
                 "The IP address to listen for call-home connections.";
              }
              leaf port {
                type inet:port-number;
                default 4334;
                description
```

```
                "The port number to listen for call-home connections.";
              }
              uses ss:ssh-client-grouping;
            }
          }
          case tls {
            if-feature tls-listen;
            container tls {
              description
                "TLS-specific listening configuration for inbound
                 connections.";
              leaf address {
                type inet:ip-address;
                description
                 "The IP address to listen for call-home connections.";
              }
              leaf port {
                type inet:port-number;
                default 4335;
                description
                 "The port number to listen for call-home connections.";
              }
              uses ts:tls-client-grouping;
            }
          }
        } // end transport
      } // end endpoint
    } // end listen

  } // end netconf-client


  grouping endpoints-container {
    description
      "This grouping is used to configure a set of NETCONF servers
       a NETCONF client may initiate connections to.";
    container endpoints {
      description
        "Container for the list of endpoints.";
      list endpoint {
        key name;
        unique "address port";
        min-elements 1;
        ordered-by user;
        description
          "A non-empty user-ordered list of endpoints for this NETCONF
           client to try to connect to.  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 client
            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 client will
            use the IANA-assigned well-known port (set via a refine
            statement when uses) if no value is specified.";
        }
      }
    }
  }
}
```

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

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

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 (host-key-type)
       |        |     |        +--:(public-key)
       |        |     |        |  +--rw public-key?
       |        |     |        |          -> /ks:keystore/keys/key/name
       |        |     |        +--:(certificate)
       |        |     |           +--rw certificate?   leafref
       |        |     |                     {sshcom:ssh-x509-certs}?
       |        |     +--rw client-cert-auth {sshcom:ssh-x509-certs}?
       |        |     |  +--rw trusted-ca-certs?       leafref
       |        |     |  +--rw trusted-client-certs?   leafref
       |        |     +--rw transport-params
       |        |             {ssh-server-transport-params-config}?
       |        |        +--rw host-key
       |        |        |  +--rw host-key-alg*   identityref
       |        |        +--rw key-exchange
       |        |        |  +--rw key-exchange-alg*   identityref
       |        |        +--rw encryption
       |        |        |  +--rw encryption-alg*   identityref
       |        |        +--rw mac
       |        |        |  +--rw mac-alg*   identityref
       |        |        +--rw compression
       |        |           +--rw compression-alg*   identityref
       |        +--:(tls) {tls-listen}?
       |           +--rw tls
       |              +--rw address?        inet:ip-address
       |              +--rw port?           inet:port-number
       |              +--rw certificates
```

```
|              |  +--rw certificate* [name]
|              |     +--rw name    leafref
|              +--rw client-auth
|              |  +--rw trusted-ca-certs?       leafref
|              |  +--rw trusted-client-certs?   leafref
|              |  +--rw cert-maps
|              |     +--rw cert-to-name* [id]
|              |        +--rw id             uint32
|              |        +--rw fingerprint    x509c2n:tls-fingerprint
|              |        +--rw map-type       identityref
|              |        +--rw name           string
|              +--rw hello-params
|                       {tls-server-hello-params-config}?
|                 +--rw tls-versions
|                 |  +--rw tls-version*   identityref
|                 +--rw cipher-suites
|                    +--rw cipher-suite*   identityref
+--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 (host-key-type)
      |  |     |        +--:(public-key)
      |  |     |        |  +--rw public-key?
      |  |     |        |          -> /ks:keystore/keys/key/name
      |  |     |        +--:(certificate)
      |  |     |           +--rw certificate?   leafref
      |  |     |                    {sshcom:ssh-x509-certs}?
      |  |     +--rw client-cert-auth {sshcom:ssh-x509-certs}?
      |  |     |  +--rw trusted-ca-certs?       leafref
      |  |     |  +--rw trusted-client-certs?   leafref
      |  |     +--rw transport-params
      |  |              {ssh-server-transport-params-config}?
      |  |        +--rw host-key
      |  |        |  +--rw host-key-alg*   identityref
      |  |        +--rw key-exchange
      |  |        |  +--rw key-exchange-alg*   identityref
      |  |        +--rw encryption
```

```
|  |        |  +--rw encryption-alg*   identityref
|  |        +--rw mac
|  |        |  +--rw mac-alg*   identityref
|  |        +--rw compression
|  |           +--rw compression-alg*   identityref
|  +--:(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    leafref
|        +--rw client-auth
|        |  +--rw trusted-ca-certs?       leafref
|        |  +--rw trusted-client-certs?   leafref
|        |  +--rw cert-maps
|        |     +--rw cert-to-name* [id]
|        |        +--rw id              uint32
|        |        +--rw fingerprint    x509c2n:tls-fingerprint
|        |        +--rw map-type       identityref
|        |        +--rw name           string
|        +--rw hello-params
|                 {tls-server-hello-params-config}?
|           +--rw tls-versions
|           |  +--rw tls-version*   identityref
|           +--rw cipher-suites
|              +--rw cipher-suite*   identityref
+--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 [I-D.ietf-netconf-keystore].

```
<netconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server"
  xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name">

  <!-- listening for SSH and TLS connections -->
  <listen>
    <endpoint> <!-- listening for SSH connections -->
      <name>netconf/ssh</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>
            <name>public-key</name>
            <public-key>ex-rsa-key</public-key>
          </host-key>
          <host-key>
            <name>certificate</name>
            <certificate>builtin-idevid-cert</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>
    <endpoint> <!-- listening for TLS sessions -->
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>
            <name>tls-ec-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>
```

```
          <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 an SSH and TLS based NETCONF clients -->
  <call-home>
    <netconf-client> <!-- SSH-based 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>
            <name>certificate</name>
            <certificate>builtin-idevid-cert</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>
    <netconf-client> <!-- TLS-based 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>
            <name>tls-ec-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>
          <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@2017-03-13.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 2017-03-13; // stable grouping definitions
    reference
      "RFC YYYY: SSH Client and Server Models";
  }

  import ietf-tls-server {
    prefix ts;
    revision-date 2017-03-13; // 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>

    Author:   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 "2017-03-13" {
    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 8071: 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 8071: 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 8071: 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.";
        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.";
              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;
                default 830;
                description
                 "The local port number on this interface the SSH server
                  listens on.";
              }
              uses ss:ssh-server-grouping;
            }
          }
          case tls {
            if-feature tls-listen;
            container tls {
              description
                "TLS-specific listening configuration for inbound
                 connections.";
              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;
                default 6513;
                description
                 "The local port number on this interface the TLS server
                  listens on.";
              }
              uses ts:tls-server-grouping {
                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: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: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 8071: 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 to configure a set of NETCONF clients
       a NETCONF server may initiate call-home connections to.";
    container endpoints {
      description
        "Container for the list of endpoints.";
      list endpoint {
        key name;
        unique "address port";
        min-elements 1;
        ordered-by user;
        description
          "A non-empty user-ordered list of endpoints for this NETCONF
           server to try to connect to.  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 (set via a refine
            statement when uses) 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 ([RFC8071]), 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).

The YANG module defined in this document uses groupings defined in [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-tls-client-server]. Please see the Security Considerations section in those documents for concerns related those groupings.

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

6. IANA Considerations

6.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. 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 [RFC7950]. Following the format in [RFC7950], 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

[I-D.ietf-netconf-keystore]
Watsen, K. and G. Wu, "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), October 2016.

[I-D.ietf-netconf-ssh-client-server]
Watsen, K. and G. Wu, "SSH Client and Server Models", draft-ietf-netconf-ssh-client-server-01 (work in progress), November 2016.

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-01 (work in progress), November 2016.

[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>.

[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>.

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

8.2. Informative References

[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>.

[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>.

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

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.

A.2. 00 to 01

- Renamed "keychain" to "keystore".

A.3. 01 to 02

- Added to ietf-netconf-client ability to connected to a cluster of endpoints, including a reconnection-strategy.
- Added to ietf-netconf-client the ability to configure connection-type and also keep-alive strategy.
- Updated both modules to accomodate new groupings in the ssh/tls drafts.

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
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

A. Gonzalez Prieto
Cisco Systems
A. Clemm
Sympotech
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
S. Chisholm
Ciena
H. Trevino
Cisco Systems
October 31, 2016

NETCONF Support for Event Notifications
draft-ietf-netconf-netconf-event-notifications-01

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 discusses 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 May 4, 2017.

Table of Contents

1. Introduction 3
1.1. Terminology 4
1.2. Solution Overview 5
2. Solution 5
2.1. Event Streams 6
2.2. Event Stream Discovery 6
2.3. Default Event Stream 9
2.4. Creating a Subscription 9
2.5. Establishing a Subscription 11
2.6. Modifying a Subscription 16
2.7. Deleting a Subscription 21
2.8. Configured Subscriptions 24
2.9. Event (Data Plane) Notifications 33
2.10. Control Plane Notifications 35
3. Backwards Compatibility 44
3.1. Capabilities 44
3.2. Stream Discovery 45
4. Security Considerations 45
5. Acknowledgments 46
6. References 46
6.1. Normative References 46
6.2. Informative References 47
Appendix A. Issues that are currently being worked 47
Appendix B. Changes between revisions 47
B.1. v00 to v01 47
Authors' Addresses 47

1. Introduction

[RFC6241] can be conceptually partitioned into four layers:

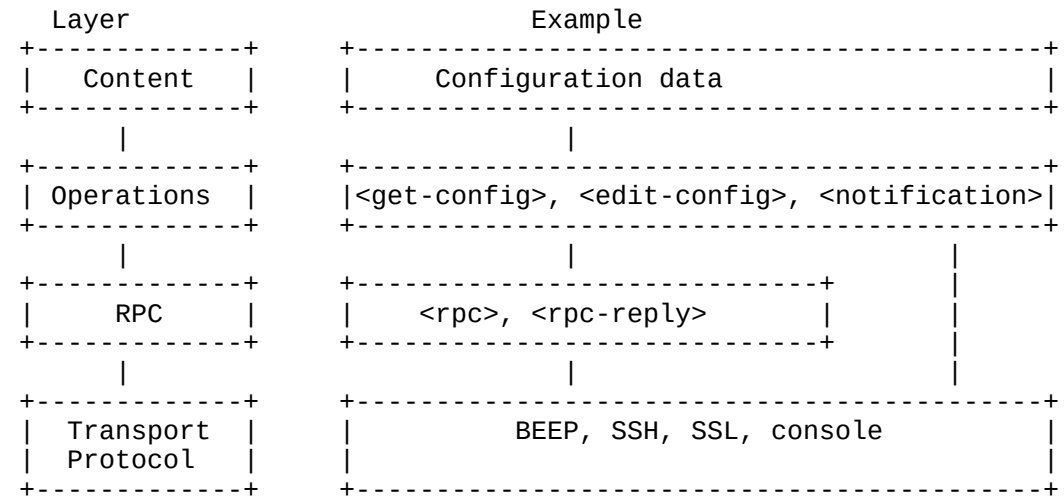


Figure 1: NETCONF layer architecture

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 include the ability to terminate subscriptions without terminating the client session, to modify existing subscriptions, and to have multiple subscriptions on a NETCONF session. [RFC5277] clients that do not require these enhancements are not affected by them.

[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. NETCONF

The following terms are defined in [RFC6241] :

- Client
- Server
- Operation
- RPC: remote procedure call

1.1.2. Event Notifications

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

- Event
- Event notification
- Stream (also referred to as "event stream")
- Subscriber

- Publisher
- Receiver
- Subscription
- Filter
- Dynamic subscription
- Configured subscription

Note that a publisher in [event-notifications] corresponds to a server in [RFC6241]. Similarly, a subscribers corresponds to a client. A receiver is also a client. In the remainder of this document, we will use the terminology in [RFC6241].

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

This operation was fully defined in [RFC5277].

2.4.1. Usage Example

The following demonstrates dynamically creating a subscription.

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

Figure 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 operation 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

This operation is defined in [event-notifications].

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 [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 subscription establishments over a single NETCONF session. In contrast, [RFC5277] mandated servers to return an error when a create-subscription was sent while a subscription was active on that session. Note that servers are not required to support multiple create-subscription over a single session, but they MUST support multiple establish-suscription over one session.

2.5.5. Message Flow Examples

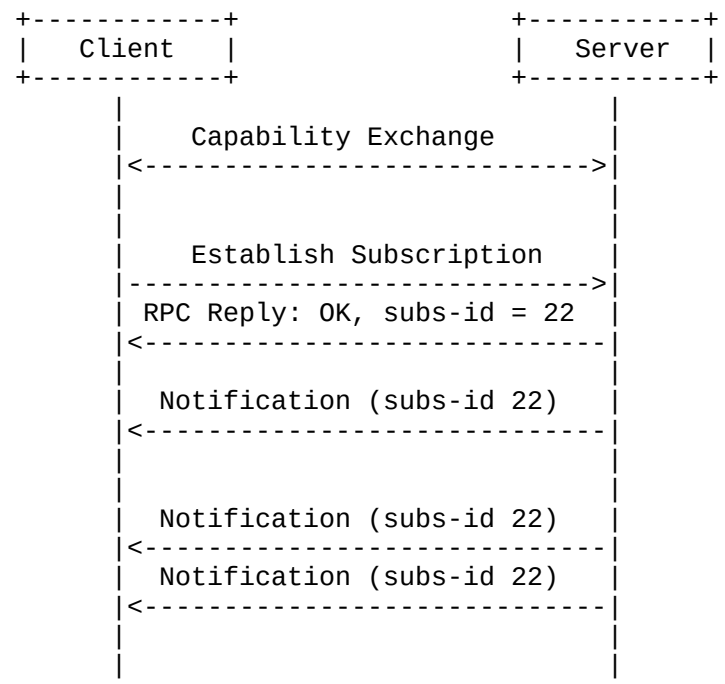


Figure 16: Message flow for subscription establishment

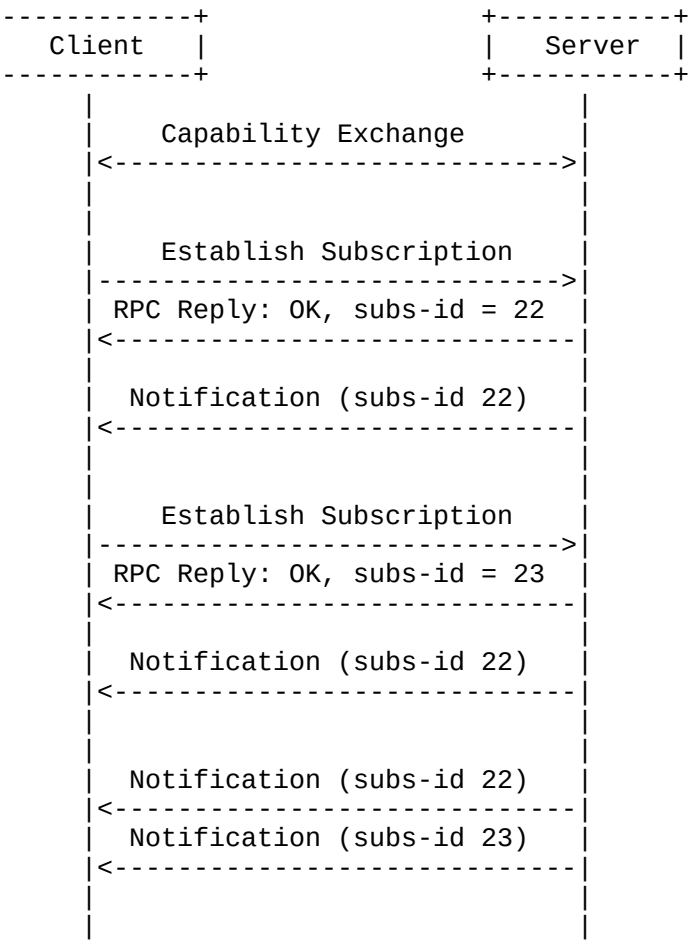


Figure 17: Message Flow for multiple subscription establishments over a single session

2.6. Modifying a Subscription

This operation is defined in [event-notifications].

2.6.1. Usage Example

The following demonstrates modifying a subscription. Consider a subscription from [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 in 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

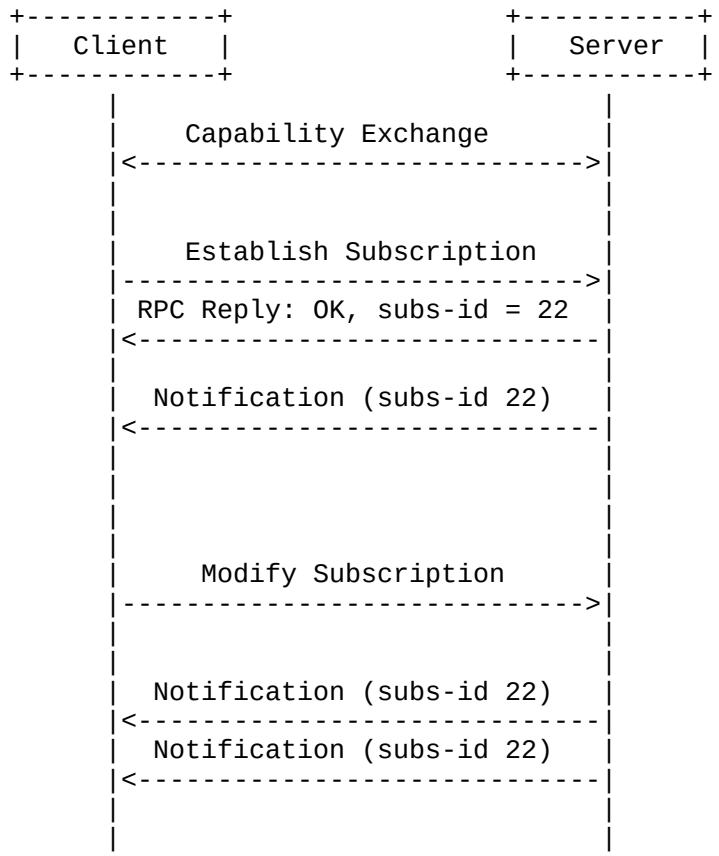


Figure 22: Message flow for subscription modification

2.7. Deleting a Subscription

This operation is defined in [event-notifications].

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

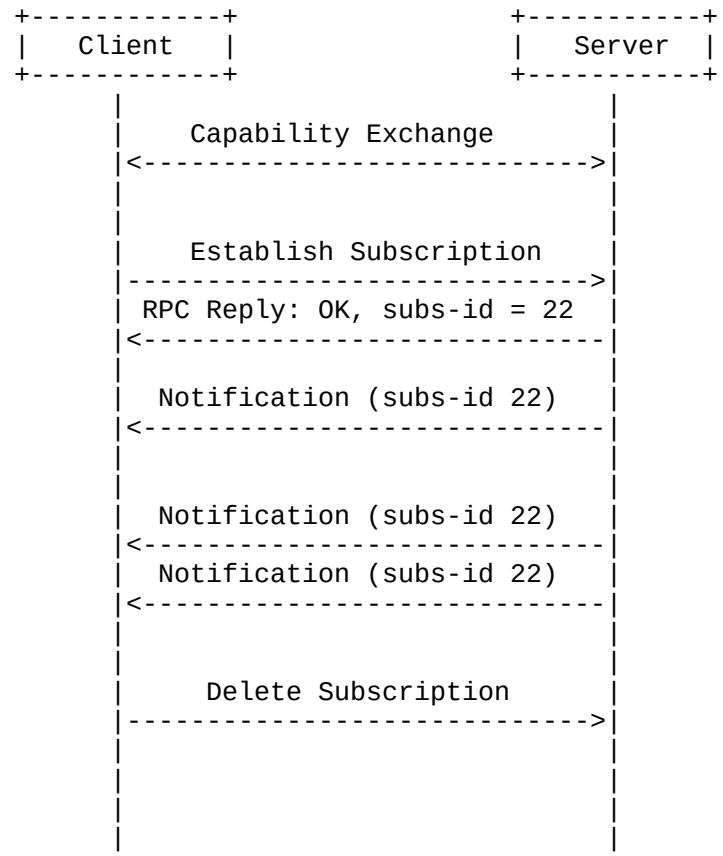


Figure 26: Message flow for subscription deletion

2.8. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface. Configured subscriptions do not support negotiation.

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

Configured susbscriptions are supported by NETCONF servers using NETCONF Call Home [call-home]

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

2.8.1. Call Home for Configured Subscriptions

Configured subscriptions are established, modified, and deleted using configuration operations against the top-level subtree subscription-config. Once the configuration is set, the server initiates a Call Home to each of the receivers in the subscription on the address and port specified. Once the NETCONF session between the server and the receiver is established, the server will issue a "subscription-started" notification. After that, the server will send notifications to the receiver as per the subscription notification.

Note that the server assumes the receiver is aware that calls on the configured port are intended only for pushing notifications. It also assumes that the receiver is ready to accept notifications on the session created as part of the Call Home as soon as the NETCONF session is established. This may require coordination between the client that configures the subscription and the clients for which the notifications are intended. This coordination is out of the scope of this document.

2.8.2. 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

2.8.2.1. Message Flow Example

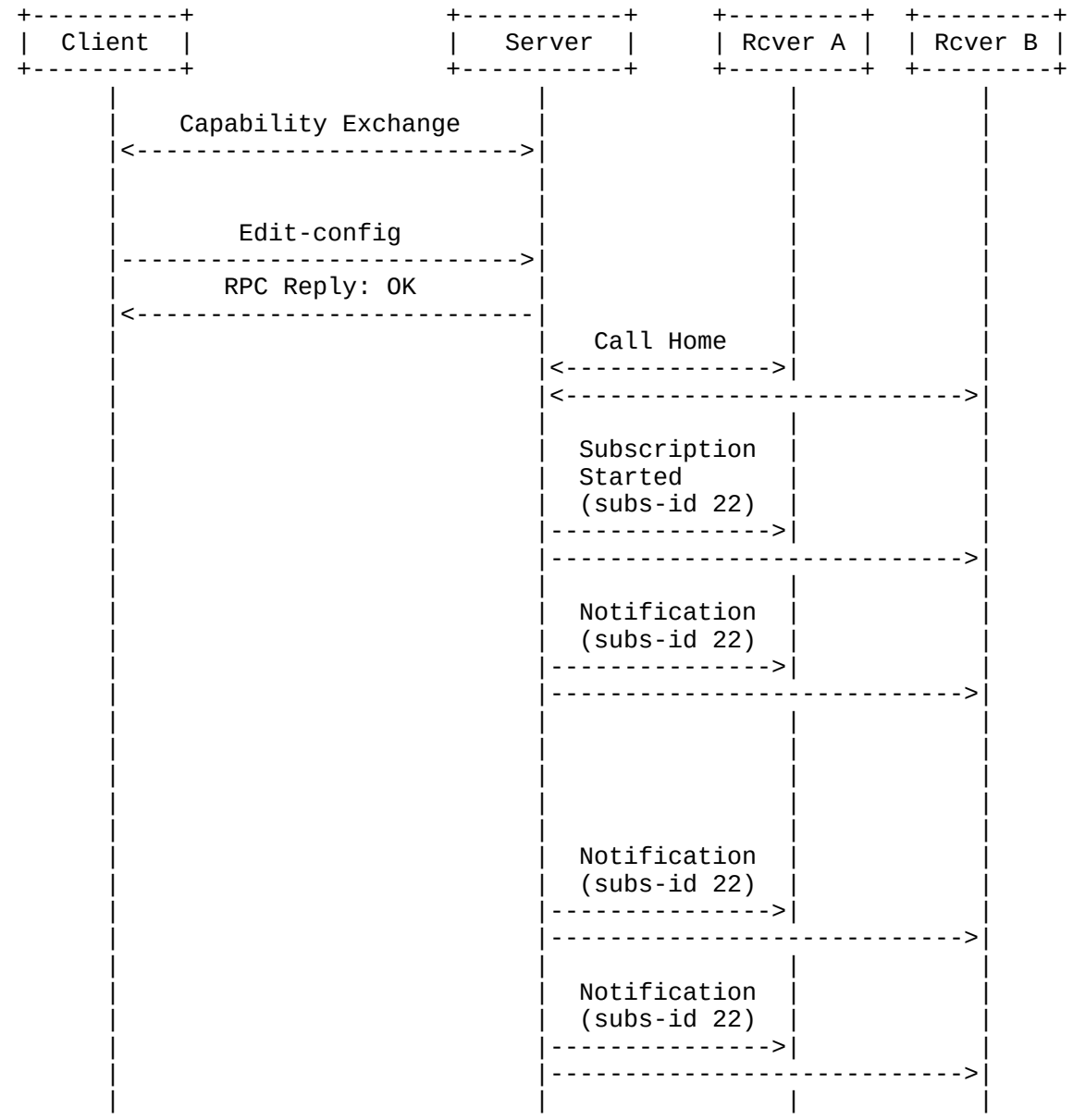


Figure 30: Message flow for subscription establishment (configured subscription)

2.8.3. 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

2.8.3.1. Message Flow Example

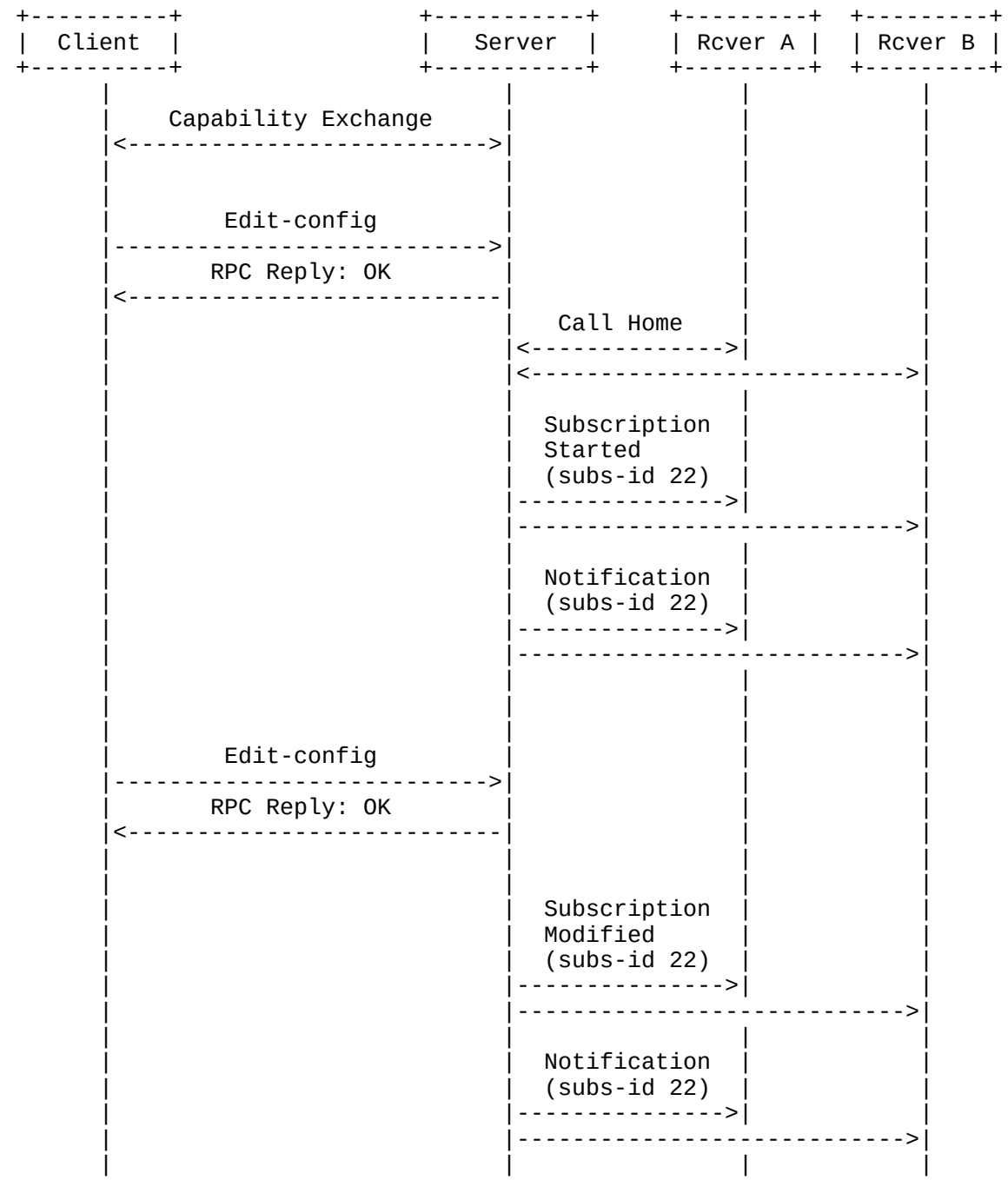


Figure 33: Message flow for subscription modification (configured subscription)

2.8.4. Deleting a Configured Subscription

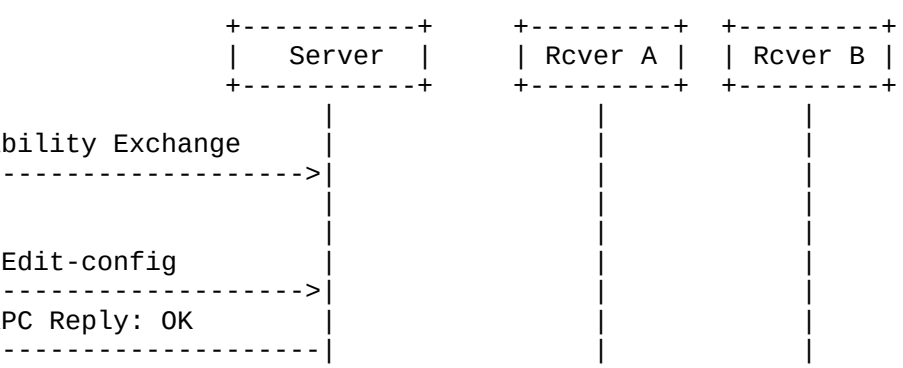
Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example:

```
<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

2.8.4.1. Message Flow Example



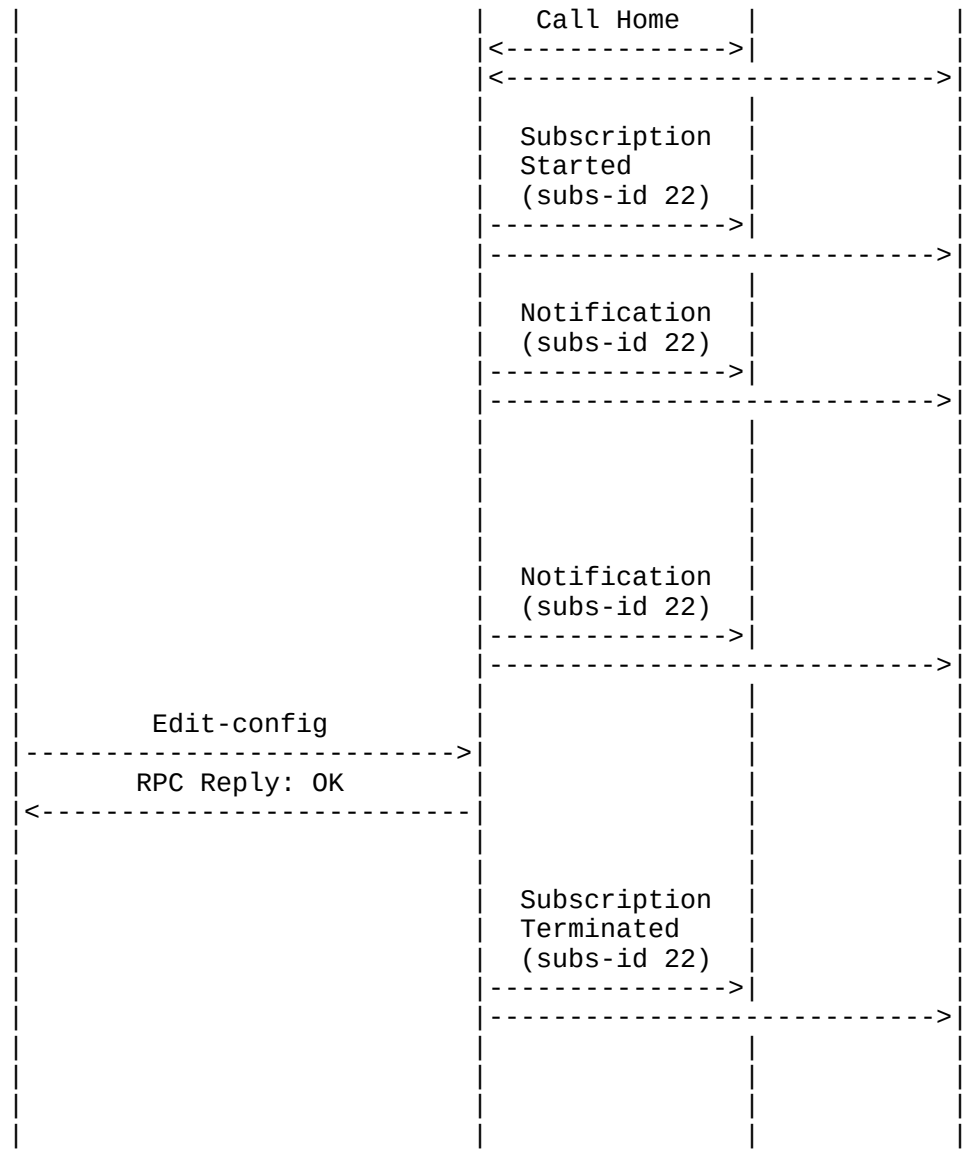


Figure 35: Message flow for subscription deletion (configured subscription)

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 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 notification-specific content:

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

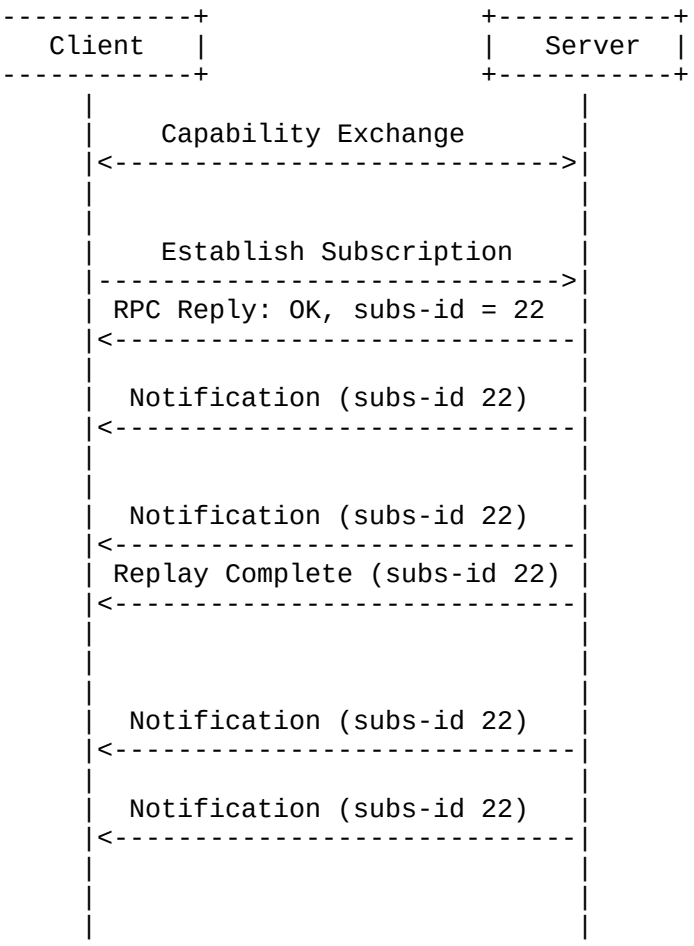


Figure 41: replayComplete notification

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

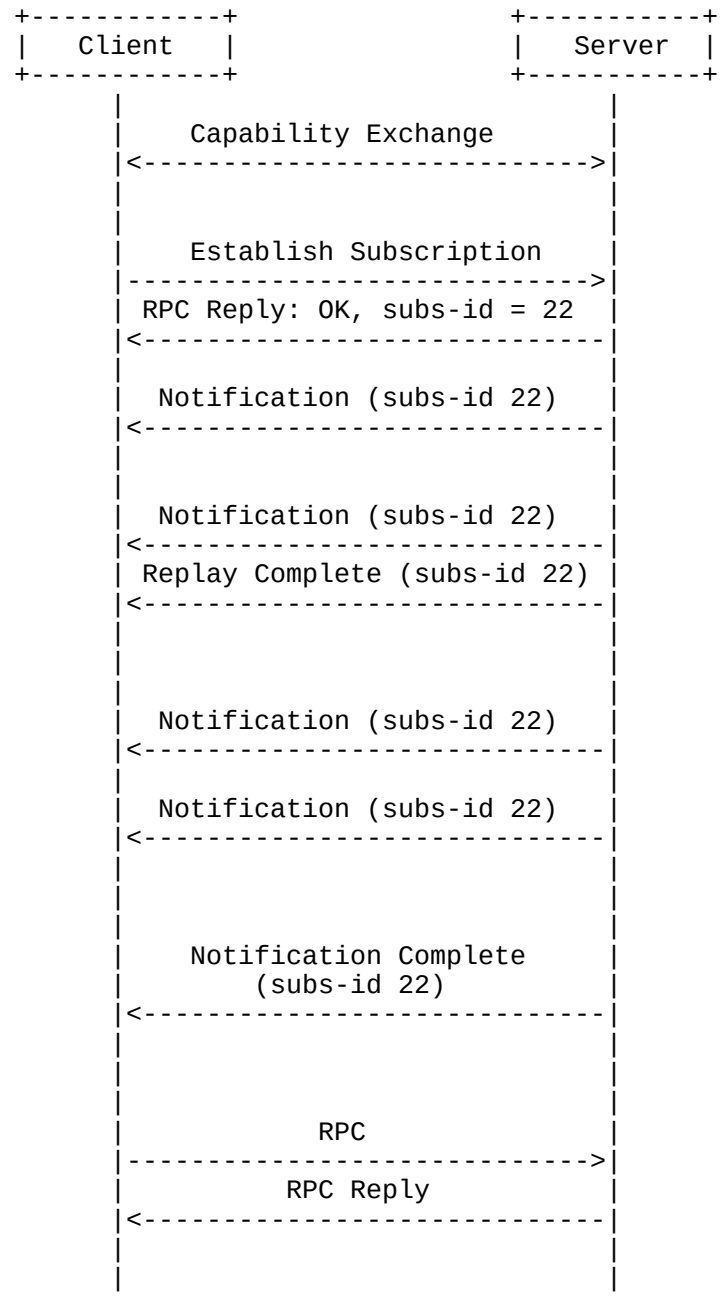


Figure 43: notificationComplete notification

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

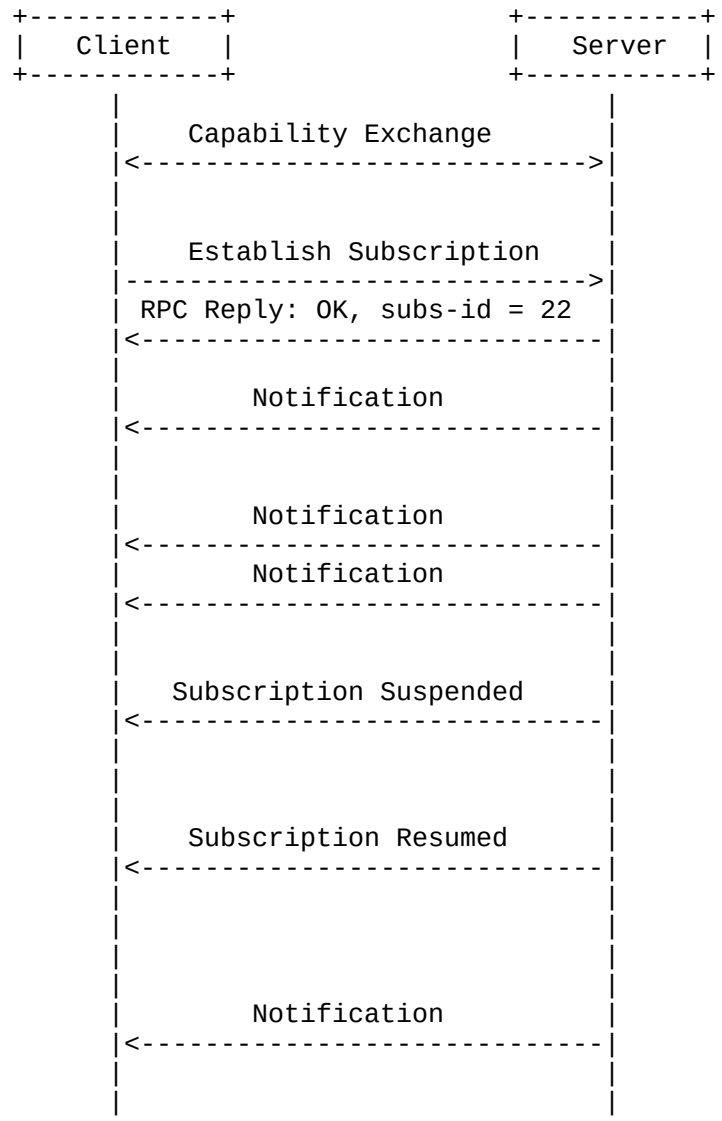
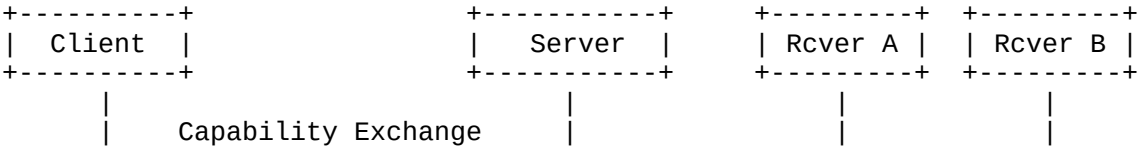


Figure 50: subscription-suspended and Resumed Notifications



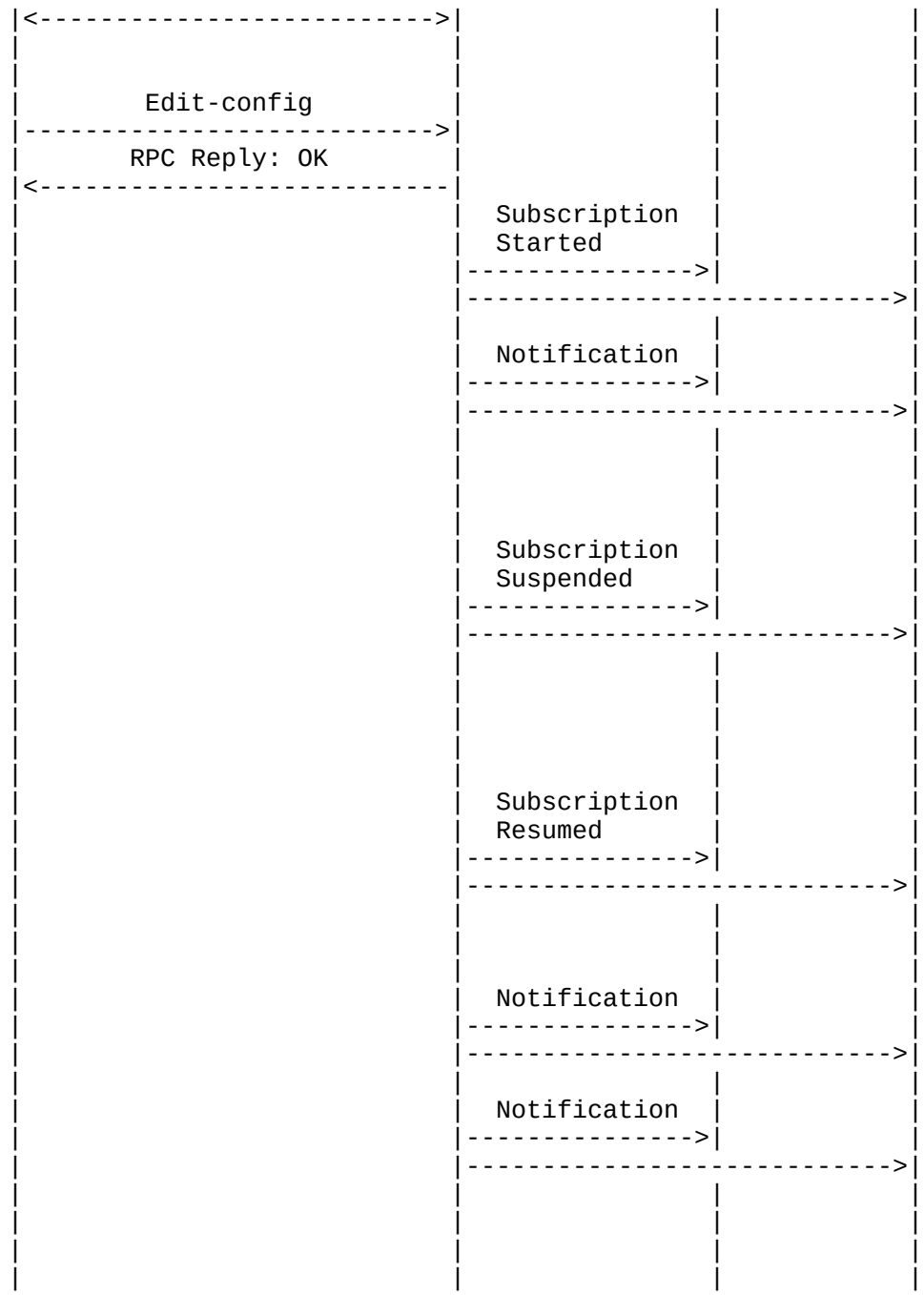

Edit-config
RPC Reply: OK
Subscription Started
Notification
Subscription Suspended
Subscription Resumed
Notification
Notification

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 [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

[call-home]
Watsen, K., "NETCONF Call Home and RESTCONF Call Home", December 2015, <https://datatracker.ietf.org/doc/draft-ietf-netconf-call-home/>.

[event-notifications]
Clemm, A., Gonzalez Prieto, A., Voit, Eric., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "Subscribing to Event Notifications", June 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-rfc5277bis/>.

[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 that are currently being worked

(To be removed by RFC editor prior to publication)

- NT1 - Express filter in JSON should be documented.

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

B.1. v00 to v01

- D1 - Added Call Home in solution for configured subscriptions.
- D2 - Clarified support for multiple subscription on a single session. No need to support multiple create-subscription.
- D3 - Added mapping between terminology in [yang-push] and [RFC6241] (the one followed in this document).
- D4 - Editorial improvements.

Authors' Addresses

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Alexander Clemm
Sympotech

Email: alex@sympotech.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: September 14, 2017

K. Watsen
Juniper Networks
J. Schoenwaelder
Jacobs University Bremen

March 13, 2017

RESTCONF Client and Server Models
draft-ietf-netconf-restconf-client-server-02

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:

- I-D.ietf-netconf-keystore
- I-D.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
- "ZZZZ" --> the assigned RFC value for I-D.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:

- "2017-03-13" --> 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 3
  1.1. Terminology 3
  1.2. Tree Diagrams 3
2. The RESTCONF Client Model 4
  2.1. Tree Diagram 4
  2.2. Example Usage 6
  2.3. YANG Model 8
3. The RESTCONF Server Model 16
  3.1. Tree Diagram 16
  3.2. Example Usage 18

3.3. YANG Model 20
4. Security Considerations 29
5. IANA Considerations 30
5.1. The IETF XML Registry 30
5.2. The YANG Module Names Registry 30
6. Acknowledgements 30
7. References 31
7.1. Normative References 31
7.2. Informative References 31
Appendix A. Change Log 33
A.1. server-model-09 to 00 33
A.2. 00 to 01 33
A.3. 01 to 02 33
Appendix B. Open Issues 33
Authors' Addresses 33

1. Introduction

This document defines two YANG [RFC7950] modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server [RFC8040]. Both modules support the TLS [RFC5246] transport protocol with both standard RESTCONF and RESTCONF Call Home connections [RFC8071].

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.

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- o 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 the TLS transport protocol using the TLS client groupings defined in [I-D.ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

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 {initiate}?
       |  +--rw restconf-server* [name]
       |     +--rw name                  string
       |     +--rw (transport)
       |     |  +--:(tls) {tls-initiate}?
       |     |     +--rw tls
       |     |        +--rw endpoints
       |     |        |  +--rw endpoint* [name]
       |     |        |     +--rw name       string
       |     |        |     +--rw address    inet:host
       |     |        |     +--rw port?      inet:port-number
       |     |        +--rw server-auth
       |     |        |  +--rw trusted-ca-certs?       leafref
       |     |        |  +--rw trusted-server-certs?   leafref
       |     |        +--rw client-auth
       |     |        |  +--rw (auth-type)?
       |     |        |     +--:(certificate)
       |     |        |        +--rw certificate?   leafref
       |     |        +--rw hello-params
```

```
|     |                {tls-client-hello-params-config}?
|     |           +--rw tls-versions
|     |           |  +--rw tls-version*   identityref
|     |           +--rw cipher-suites
|     |              +--rw cipher-suite*   identityref
|     +--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
+--rw listen {listen}?
   +--rw max-sessions?   uint16
   +--rw idle-timeout?   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 server-auth
               |  +--rw trusted-ca-certs?       leafref
               |  +--rw trusted-server-certs?   leafref
               +--rw client-auth
               |  +--rw (auth-type)?
               |     +--:(certificate)
               |        +--rw certificate?   leafref
               +--rw hello-params
                       {tls-client-hello-params-config}?
                  +--rw tls-versions
                  |  +--rw tls-version*   identityref
                  +--rw cipher-suites
                     +--rw cipher-suite*   identityref
```

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 [I-D.ietf-netconf-keystore].

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

  <!-- RESTCONF servers to initiate connections to -->
  <initiate>
    <restconf-server>
      <name>corp-fw1</name>
      <tls>
        <endpoints>
          <endpoint>
            <name>corp-fw1.example.com</name>
            <address>corp-fw1.example.com</address>
          </endpoint>
          <endpoint>
            <name>corp-fw2.example.com</name>
            <address>corp-fw2.example.com</address>
          </endpoint>
        </endpoints>
        <server-auth>
          <trusted-server-certs>deployment-specific-ca-certs</trusted-server-cer
ts>
        </server-auth>
        <client-auth>
          <certificate>tls-ec-cert</certificate>
        </client-auth>
      </tls>
    </restconf-server>
  </initiate>

  <!-- endpoints to listen for RESTCONF Call Home connections on -->
  <listen>
    <endpoint>
      <name>Intranet-facing listener</name>
      <tls>
        <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>
        </server-auth>
        <client-auth>
          <certificate>tls-ec-cert</certificate>
        </client-auth>
      </tls>
    </endpoint>
  </listen>
</restconf-client>
```

2.3. YANG Model

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

<CODE BEGINS> file "ietf-restconf-client@2017-03-13.yang"

```
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-tls-client {
    prefix ts;
    revision-date 2017-03-13; // 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/restconf/>
    WG List:  <mailto:restconf@ietf.org>

    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 RESTCONF clients.

    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 "2017-03-13" {
    description
     "Initial version";
    reference
     "RFC XXXX: RESTCONF Client and Server Models";
  }

  // Features

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

  feature tls-initiate {
    description
     "The 'tls-initiate' feature indicates that the RESTCONF client
      supports initiating TLS connections to RESTCONF servers.";
    reference
     "RFC 8040: RESTCONF Protocol";
  }

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

  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 8071: NETCONF Call Home and RESTCONF Call Home";
```

```
  }

  container restconf-client {
    description
      "Top-level container for RESTCONF client configuration.";

    container initiate {
      if-feature initiate;
      description
        "Configures client initiating underlying TCP connections.";
      list restconf-server {
        key name;
        description
          "List of RESTCONF servers the RESTCONF client is to initiate
           connections to.";
        leaf name {
          type string;
          description
            "An arbitrary name for the RESTCONF server.";
        }
        choice transport {
          mandatory true;
          description
            "Selects between available transports.";

          case tls {
            if-feature tls-initiate;
            container tls {
              description
                "Specifies TLS-specific transport configuration.";
              uses endpoints-container {
                refine endpoints/endpoint/port {
                  default 443;
                }
              }
              uses ts:tls-client-grouping;
            }
          } // end tls

        } // end transport

        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 RESTCONF
                server. If the connection goes down, immediately
                start trying to reconnect to it, using the
                reconnection strategy.

                This connection type minimizes any RESTCONF server
                to RESTCONF client 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 RESTCONF session may remain idle. A RESTCONF
                   session will be dropped if it is idle for an
                   interval longer than this number of seconds.
                   If set to zero, then the client 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 server.  An
                   unresponsive SSH/TLS server will be dropped after
                   approximately max-attempts * max-wait seconds.";
                reference
                  "RFC 8071: 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
                    server, a SSH/TLS-level message will be sent
                    to test the aliveness of the SSH/TLS server.";
                }
                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 server before assuming the SSH/TLS
                      server is no longer alive.";
                  }
                }
              }
            }
            case periodic-connection {
              container periodic {
                presence true;
                description
                 "Periodically connect to the RESTCONF server, so that
                  the RESTCONF server may deliver messages pending for
                  the RESTCONF client.  The RESTCONF server must close
                  the connection when it is ready to release it. Once
                  the connection has been closed, the RESTCONF client
                  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 RESTCONF session may remain idle. A RESTCONF
                     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
                    RESTCONF client will wait before re-establishing
                    a connection to the RESTCONF server.  The RESTCONF
                    client may initiate a connection before this
                    time if desired (e.g., to set configuration).";
                }
              }
```

```
            }
          }
        }
        container reconnect-strategy {
          description
           "The reconnection strategy directs how a RESTCONF client
            reconnects to a RESTCONF server, after discovering its
            connection to the server has dropped, even if due to a
            reboot.  The RESTCONF client 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
                   clients SHOULD be able to remember the last
                   endpoint connected to across reboots.";
              }
            }
            default first-listed;
            description
             "Specifies which of the RESTCONF server's endpoints the
              RESTCONF client should start with when trying to connect
              to the RESTCONF server.";
          }
          leaf max-attempts {
            type uint8 {
              range "1..max";
            }
            default 3;
            description
             "Specifies the number times the RESTCONF client tries to
              connect to a specific endpoint before moving on to the
              next endpoint in the list (round robin).";
          }
        }
      } // end restconf-server
    } // 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 RESTCONF
           session may remain idle. A RESTCONF 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 RESTCONF connections.";
        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.";
              leaf address {
                type inet:ip-address;
```

```
                description
                 "The IP address to listen for call-home connections.";
               }
               leaf port {
                 type inet:port-number;
                 default 4336;
                 description
                  "The port number to listen for call-home connections.";
               }
               uses ts:tls-client-grouping;
             }
           }
         } // end transport
       } // end endpoint
     } // end listen

   } // end restconf-client


   grouping endpoints-container {
     description
       "This grouping is used to configure a set of RESTCONF servers
        a RESTCONF client may initiate connections to.";
     container endpoints {
       description
         "Container for the list of endpoints.";
       list endpoint {
         key name;
         unique "address port";
         min-elements 1;
         ordered-by user;
         description
           "A non-empty user-ordered list of endpoints for this RESTCONF
            client to try to connect to.  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 client
             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 client will
            use the IANA-assigned well-known port (set via a refine
            statement when uses) if no value is specified.";
        }
      }
    }
  }
}
```

<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 transport protocol using the TLS server groupings defined in [I-D.ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

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    leafref
|              +--rw client-auth
|              |  +--rw trusted-ca-certs?       leafref
|              |  +--rw trusted-client-certs?   leafref
|              |  +--rw cert-maps
|              |     +--rw cert-to-name* [id]
|              |        +--rw id             uint32
|              |        +--rw fingerprint    x509c2n:tls-fingerprint
|              |        +--rw map-type       identityref
|              |        +--rw name           string
|              +--rw hello-params
|                       {tls-server-hello-params-config}?
|                 +--rw tls-versions
|                 |  +--rw tls-version*   identityref
|                 +--rw cipher-suites
|                    +--rw cipher-suite*   identityref
|
+--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    leafref
      |        +--rw client-auth
      |        |  +--rw trusted-ca-certs?       leafref
      |        |  +--rw trusted-client-certs?   leafref
      |        |  +--rw cert-maps
      |        |     +--rw cert-to-name* [id]
      |        |        +--rw id             uint32
      |        |        +--rw fingerprint    x509c2n:tls-fingerprint
      |        |        +--rw map-type       identityref
      |        |        +--rw name           string
      |        +--rw hello-params
      |                 {tls-server-hello-params-config}?
      |           +--rw tls-versions
      |           |  +--rw tls-version*   identityref
      |           +--rw cipher-suites
      |              +--rw cipher-suite*   identityref
      +--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 [I-D.ietf-netconf-keystore].

```
<restconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server"
  xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name">

  <!-- listening for TLS (HTTPS) connections -->
  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>
            <name>tls-ec-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>
          <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>
            <name>tls-ec-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>
          <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@2017-03-13.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 2017-03-13; // 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 "2017-03-13" {
    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 8071: 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 8071: 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.";
        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.";
              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;
                default 443;
                description
                 "The local port number on this interface the TLS server
                  listens on.";
              }
              uses ts:tls-server-grouping {
                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: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 8071: 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;
        unique "address port";
        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

The YANG module defined in this document uses a grouping defined in [I-D.ietf-netconf-tls-client-server]. Please see the Security Considerations section in that document for concerns related that grouping.

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

5. IANA Considerations

5.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. 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 [RFC7950]. Following the format in [RFC7950], 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

[I-D.ietf-netconf-keystore]
Watsen, K. and G. Wu, "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), October 2016.

[I-D.ietf-netconf-tls-client-server]
Watsen, K., "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-01 (work in progress), November 2016.

[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>.

[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>.

[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>.

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

7.2. Informative References

[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>.

[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>.

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

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.

A.2. 00 to 01

- Renamed "keychain" to "keystore".

A.3. 01 to 02

- Filled in previously missing 'ietf-restconf-client' module.
- Updated the ietf-restconf-server module to accomodate new grouping 'ietf-tls-server-grouping'.

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
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

E. Voit
A. Gonzalez Prieto
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Clemm
Huawei
A. Bierman
YumaWorks
March 13, 2017

Restconf and HTTP Transport for Event Notifications
draft-ietf-netconf-restconf-notif-02

Abstract

This document defines Restconf, HTTP2, and HTTP1.1 bindings for the transport of Subscription requests and corresponding push updates. Being subscribed may be either Event Notifications or objects or subtress of YANG Datastores.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Solution 3
3.1. Dynamic YANG Subscription with RESTCONF control 3
3.2. Subscription Multiplexing 6
4. Encoded Subscription and Event Notification Examples 7
4.1. Restconf Subscription and Events over HTTP1.1 7
4.2. Event Notification over HTTP2 12
5. Security Considerations 12
6. Acknowledgments 13
7. References 13
7.1. Normative References 13
7.2. Informative References 14
Appendix A. End-to-End Deployment Guidance 14
A.1. Call Home 14
A.2. TLS Heartbeat 15
Appendix B. Issues being worked and resolved 15
B.1. Unresolved Issues 15
Appendix C. Changes between revisions 15
Authors' Addresses 16

1. Introduction

Mechanisms to support Event subscription and push are defined in [sn]. Enhancements to [sn] which enable YANG Datastore subscription and push are defined in [yang-push]. This document provides a transport specification for these protocols over Restconf and HTTP. Driving these requirements is [RFC7923].

The streaming of Subscription Event Notifications has synergies with HTTP2 streams. Benefits which can be realized when transporting events directly HTTP2 [RFC7540] include:

- Elimination of head-of-line blocking
- Weighting and proportional dequeuing of Events from different subscriptions
- Explicit precedence in subscriptions so that events from one subscription must be sent before another dequeues

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].

The following terms use the defintions from [sn]: Configured Subscription, Dynamic Subscription, Event Notification, Publisher, Receiver, Subscriber, Subscription.

3. Solution

Event subscription is defined in [sn], YANG Datastore subscription is defined in [yang-push]. This section specifies transport mechanisms applicable to both.

3.1. Dynamic YANG Subscription with RESTCONF control

Dynamic Subscriptions for both [sn] and its [yang-push] augmentations are configured and managed via signaling messages transported over [RFC8040]. These interactions will be accomplished via a Restconf POST into RPCs located on the Publisher. HTTP responses codes will indicate the results of the interaction with the Publisher. An HTTP status code of 200 is the proper response to a successful <establish-subscription> RPC call. The successful <establish-subscription> will result in a HTTP message with returned subscription URI on a logically separate mechanism than was used for the original Restconf POST. This mechanism is via a parallel TCP connection in the case of HTTP 1.x, or in the case of HTTP2 via a separate HTTP stream within the HTTP connection. When a being returned by the Publisher, failure will be indicated by error codes transported in payload.

Once established, the resulting stream of Event Notifications are then delivered via SSE for HTTP1.1 and via HTTP Data for HTTP2.

3.1.1. Call Flow for HTTP2

Requests to [yang-push] augmented RPCs are sent on one or more HTTP2 streams indicated by (a) in Figure 2. Event Notifications related to a single subscription are pushed on a unique logical channel (b). In the case below, a newly established subscription has its events pushed over HTTP2 stream (7).

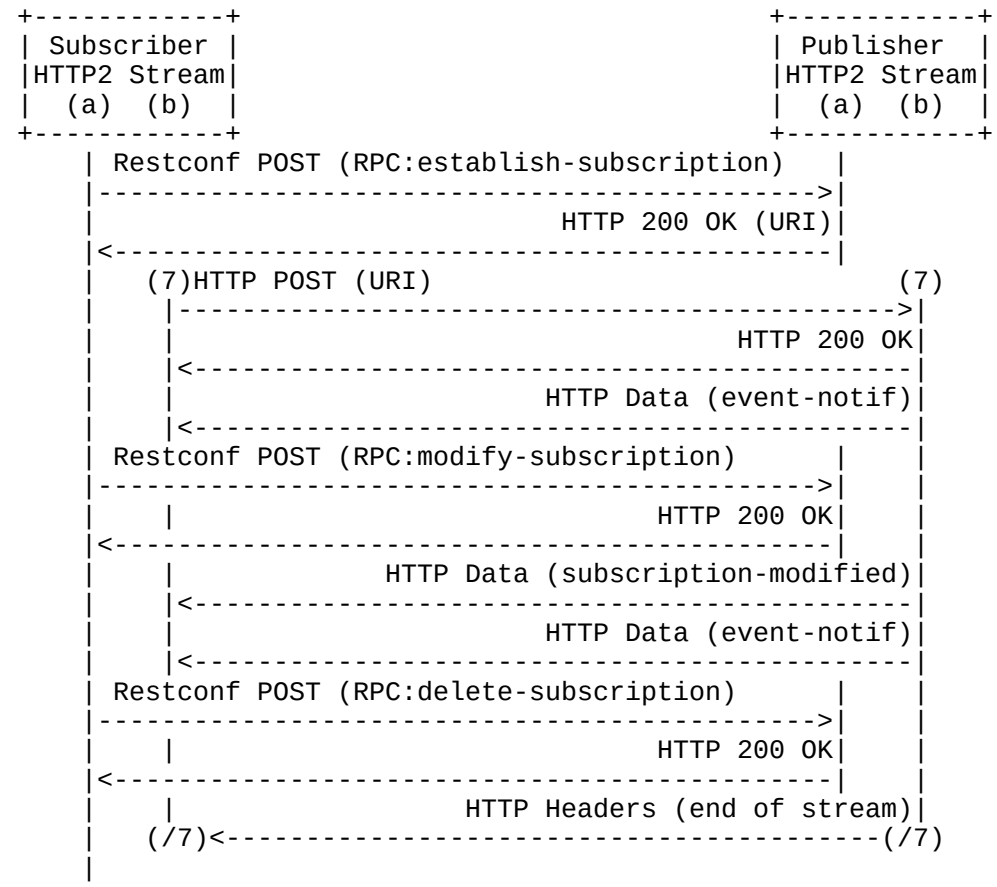


Figure 1: Dynamic with HTTP2

3.1.2. Call flow for HTTP1.1

Requests to [yang-push] RPCs are sent on the TCP connection indicated by (a). Event Notifications are pushed on a separate connection (b). This connection (b) will be used for all Event Notifications across all subscriptions.

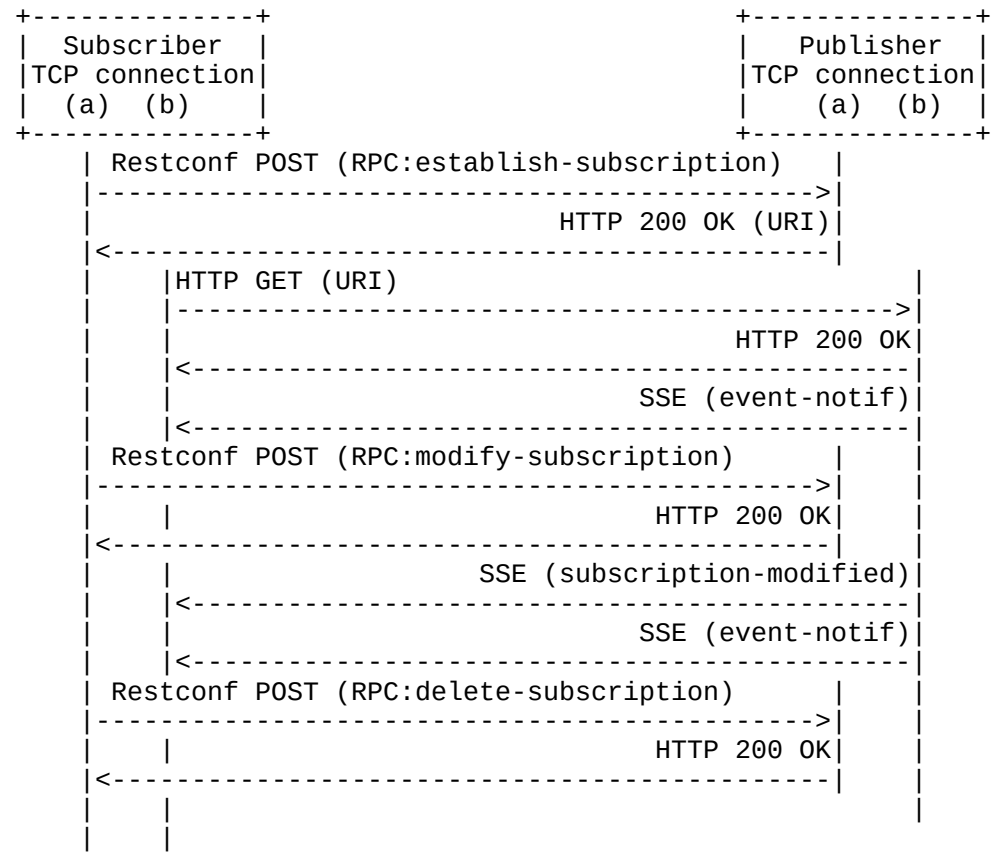


Figure 2: Dynamic with HTTP1.1

3.1.3. Configured Subscription over HTTP2

With a Configured Subscription, all information needed to establish a secure relationship with that Receiver is available 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 with any dependency on Restconf. Nor is there value which Restconf provides on top of HTTP. Therefore in place of Restconf, a TLS secured HTTP2 Client connection must be established with an HTTP2 Server located on the Receiver. Event Notifications will then be sent as part of an extended HTTP POST to the Receiver.

POST messages will be addressed to HTTP augmentation code on the Receiver capable of accepting and responding to Event Notifications. The first POST message must be a subscription-started notification. Push update notifications must not be sent until the receipt of an HTTP 200 OK for this initial notification. The 200 OK will indicate that the Receiver is ready for Event Notifications. At this point a Subscription must be allocated its own HTTP2 stream. Figure 4 depicts this message flow.

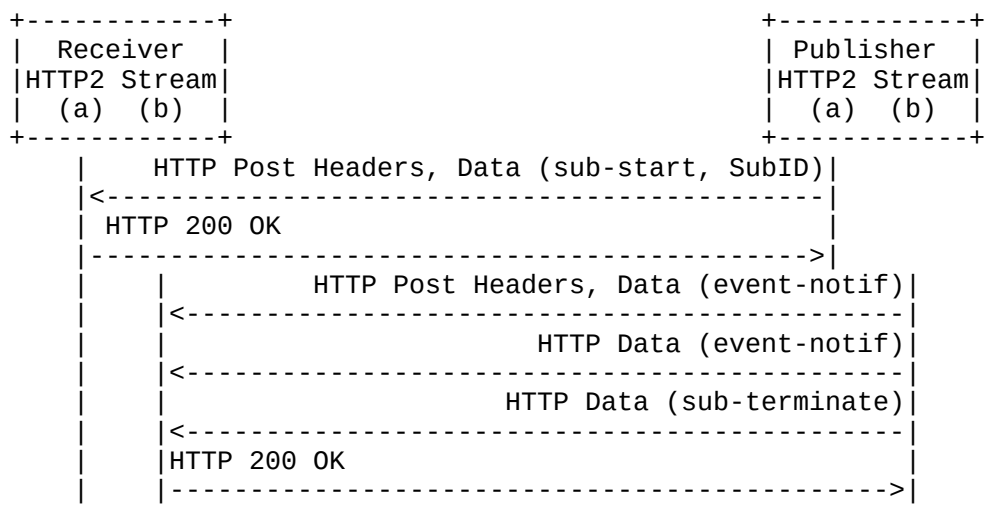


Figure 3: Configured over HTTP2

As the HTTP2 transport is available to the Receiver, the Publisher should:

- take any subscription-priority and copy it into the HTTP2 stream priority, and

- take a subscription-dependency if it has been provided and map the HTTP2 stream for the parent subscription into the HTTP2 stream dependency.

3.2. Subscription Multiplexing

It is possible that updates might be delivered in a different sequence than generated. Reasons for this might include (but are not limited to):

- replay of pushed updates

- temporary loss of transport connectivity, with update buffering and different dequeuing priorities per Subscription

- population, marshalling and bundling of independent Subscription Updates, and

Therefore each Event Notification will include a 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.

4. Encoded Subscription and Event Notification Examples

Transported updates will contain context data for one or more Event Notifications. Each transported Event Notification will contain several parameters:

4.1. Restconf Subscription and Events over HTTP1.1

Subscribers can dynamically learn whether a RESTCONF server supports various types of Event or Yang datastore subscription capabilities. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream. Some examples building upon the Call flow for HTTP1.1 from Section 3.2.2 are:

```
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 as a 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-20150203] transport strategy to push filtered Event Notifications from the Event stream.

The Publisher MUST support individual parameters within the POST request body for 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
POST /restconf/operations/ietf-event-notifications:
     establish-subscription HTTP/1.1
      Host: example.com
      Content-Type: application/yang-data+json

      {
        "ietf-event-notifications:input" : {
          ?stream?: ?push-data"
          ?period" : 5,
          "xpath-filter" : ?/ex: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>2015-03-09T19:14:56.233Z</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 [RFC7951]:

```
{
  "ietf-restconf:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.233Z",
    "datastore-push:datastore-contents": {
      "example-mod:foo": { "bar": "some_string" }
    }
  }
}
```

To modify a Subscription, the subscriber issues another POST 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:

```
POST /restconf/operations/ietf-event-notifications:
      modify-subscription HTTP/1.1
      Host: example.com
      Content-Type: application/yang-data+json

      {
        "ietf-event-notifications:input" : {
          ?subscription-id?: 100,
          ?period" : 10
        }
      }
```

To delete a Subscription, the Subscriber issues a DELETE request on the provided URI using the same subscription-id as in the original request

4.2. Event Notification over HTTP2

The basic encoding will look as below. It will consists of a JSON representation wrapped in an HTTP2 header.

```
HyperText Transfer Protocol 2
      Stream: HEADERS, Stream ID: 5
      Header: :method: POST
      Stream: HEADERS, Stream ID: 5

{
  "ietf-yangpush:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.233Z",
    "datastore-push:datastore-contents": {
      "foo": { "bar": "some_string" }
    }
  }
}
```

5. Security Considerations

Subscriptions could be used to intentionally or accidentally overload the 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 of Configured Subscriptions could be used to overwhelm a Receiver which doesn’t even support Subscriptions. There are two protections here. First 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. Second, the HTTP transport augmentation on the Receiver must send an HTTP 200 OK to a subscription started notification before the Publisher starts streaming any events.

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, HTTP2 transport such as HTTP2) should be selected.

6. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, Michael Scharf, and Guangying Zheng.

7. References

7.1. Normative References

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

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <http://www.rfc-editor.org/info/rfc6520>.

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

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

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

[sn] Voit, E., Clemm, A., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to Event Notifications", February 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>.

7.2. Informative References

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

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

[W3C-20150203] "Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", February 2015, <https://www.w3.org/TR/2015/REC-eventsource-20150203/>.

[yang-push] Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", March 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Appendix A. 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.

A.1. Call Home

Pub/Sub implementations should have the ability to transparently incorporate ’call home’ [RFC8071] so that secure TLS connections can originate from the desired device.

A.2. TLS Heartbeat

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 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 any Subscription related TCP sessions between those endpoints being torn down. The subscription can then attempt to re-establish.

Appendix B. Issues being worked and resolved

(To be removed by RFC editor prior to publication)

B.1. Unresolved Issues

GRPC compatibility 1: Mechanisms for HTTP2 to GRPC mapping need to be considered. There is a good start there as this draft only uses POST, not GET. As GET is used in RESTCONF for capabilities discovery, we have some backwards compatibility issues with existing IETF drafts. Possible options to address are (1) provide a POST method for anything done by GET in RESTCONF, (2) await support of GET by GRPC, or (3) tunnel RESTCONF's GET messages within a GRPC POST.

GRPC compatibility 2: We need to expose a method against which POST is done as events begin on a stream. See Stream 7 in figure 2. Can only send traffic to a method, not a URI. URI points to a method, not a resource.

Need to add into document examples of Event streams. Document only includes yang-push examples at this point.

We need to reference the viable encodings of notifications.

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v01 - v02

- Removed sections now redundant with [sn] and [yang-push] such as: mechanisms for subscription maintenance, terminology definitions, stream discovery.

- 3rd party subscriptions are out-of-scope.

- SSE only used with Restconf and HTTP1.1 Dynamic Subscriptions

- Timeframes for event tagging are self-defined.
- Clean-up of wording, references to terminology, section numbers.

v00 - v01

- Removed the ability for more than one subscription to go to a single HTTP2 stream.
- Updated call flows. Extensively.
- SSE only used with Restconf and HTTP1.1 Dynamic Subscriptions
- HTTP is not used to determine that a Receiver has gone silent and is not Receiving Event Notifications
- Many clean-ups of wording and terminology

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Systems

March 13, 2017

SSH Client and Server Models
draft-ietf-netconf-ssh-client-server-02

Abstract

This document defines three YANG modules: the first defines groupings for a generic SSH client, the second defines groupings for a generic SSH server, and the third defines common identities and groupings used by both the client and the 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:

- I-D.ietf-netconf-keystore

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 I-D.ietf-netconf-keystore

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

- "2017-03-13" --> 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 3
1.1. Terminology 3
1.2. Tree Diagrams 4
2. The SSH Client Model 4
2.1. Tree Diagram 4
2.2. Example Usage 5
2.3. YANG Model 6
3. The SSH Server Model 10
3.1. Tree Diagram 10
3.2. Example Usage 11
3.3. YANG Model 12
4. The SSH Common Model 15

4.1. Tree Diagram 16
4.2. Example Usage 16
4.3. YANG Model 17
5. Security Considerations 27
6. IANA Considerations 28
6.1. The IETF XML Registry 28
6.2. The YANG Module Names Registry 29
7. Acknowledgements 29
8. References 29
8.1. Normative References 29
8.2. Informative References 30
Appendix A. Change Log 32
A.1. server-model-09 to 00 32
A.2. 00 to 01 32
A.3. 01 to 02 32
Appendix B. Open Issues 32
Authors' Addresses 32

1. Introduction

This document defines three YANG [RFC7950] modules: the first defines a grouping for a generic SSH client, the second defines a grouping for a generic SSH server, and the third defines identities and groupings common to both the client and the 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 client and server YANG modules in this document each define one grouping, which is focused on just SSH-specific configuration, and specifically avoids any transport-level configuration, such as what ports to listen-on or connect-to. This enables applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the grouping for the SSH parts it provides, while adding data nodes for the TCP-level call-home configuration.

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 one YANG grouping, to just configure the SSH client omitting, for instance, any configuration for which IP address or port the client should connect to.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore 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 grouping defined in the ietf-ssh-client module. Please see Section 1.2 for tree diagram notation.

```
module: ietf-ssh-client
  groupings:
  ssh-client-grouping
      +---- server-auth
      |  +---- trusted-ssh-host-keys?
      |  |       -> /ks:keystore/trusted-host-keys/name
      |  +---- trusted-ca-certs?
      |  |       -> /ks:keystore/trusted-certificates/name
      |  |       {sshcom:ssh-x509-certs}?
      |  +---- trusted-server-certs?
      |          -> /ks:keystore/trusted-certificates/name
      |          {sshcom:ssh-x509-certs}?
      +---- client-auth
      |  +---- username?      string
      |  +---- (auth-type)?
      |     +--:(certificate)
      |     |  +---- certificate?   leafref {sshcom:ssh-x509-certs}?
      |     +--:(public-key)
      |     |  +---- public-key?    -> /ks:keystore/keys/key/name
      |     +--:(password)
      |        +---- password?      union
      +---- transport-params {ssh-client-transport-params-config}?
         +---- host-key
         |  +---- host-key-alg*   identityref
         +---- key-exchange
         |  +---- key-exchange-alg*   identityref
         +---- encryption
         |  +---- encryption-alg*   identityref
         +---- mac
         |  +---- mac-alg*   identityref
         +---- compression
            +---- compression-alg*   identityref
```

2.2. Example Usage

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

```
<!-- hypothetical example, as groupings don't have instance data -->
<ssh-client xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client">

  <!-- which host-keys will this client trust -->
  <server-auth>
    <trusted-ssh-host-keys>explicitly-trusted-ssh-host-keys</trusted-ssh-host-ke
ys>
  </server-auth>

  <!-- how this client will authenticate itself to the server -->
  <client-auth>
    <username>foobar</username>
    <public-key>ex-rsa-key</public-key>
  </client-auth>

</ssh-client>
```

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-ssh-client@2017-03-13.yang"

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

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

  import ietf-ssh-common {
    prefix sshcom;
    revision-date 2017-03-13; // stable grouping definitions
    reference
      "RFC XXXX: SSH Client and Server Models";
  }

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

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
```

```
   }

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

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

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

     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 "2017-03-13" {
     description
      "Initial version";
     reference
      "RFC XXXX: SSH Client and Server Models";
   }

   feature ssh-client-transport-params-config {
     description
       "SSH transport layer parameters are configurable on an SSH
        client.";
   }

   grouping ssh-client-grouping {
     description
```

```
      "A reusable grouping for configuring a SSH client without
       any consideration for how an underlying TCP session is
       established.";

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

      leaf trusted-ssh-host-keys {
        type leafref {
          path "/ks:keystore/ks:trusted-host-keys/ks: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 sshcom:ssh-x509-certs;
        type leafref {
          path "/ks:keystore/ks:trusted-certificates/ks: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 {
        if-feature sshcom:ssh-x509-certs;
        type leafref {
          path "/ks:keystore/ks:trusted-certificates/ks: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.";
```

```
        leaf username {
          type string;
          description
            "The username of this user.  This will be the username
             used, for instance, to log into an SSH server.";
        }

        choice auth-type {
          description
            "The authentication type.";
          leaf certificate {
            if-feature sshcom:ssh-x509-certs;
            type leafref {
              path "/ks:keystore/ks:keys/ks:key/ks:certificates/"
                   + "ks:certificate/ks:name";
            }
            description
              "A certificates to be used for user authentication.";
          }
          leaf public-key {
            type leafref {
              path "/ks:keystore/ks:keys/ks:key/ks:name";
            }
            description
              "A public keys to be used for user authentication.";
          }
          leaf password {
            nacm:default-deny-all;
            type union {
              type string;
              type enumeration {
                enum "RESTRICTED" {
                 description
                  "The private key is restricted due to access-control.";
                }
              }
            }
            description
              "A password to be used for user authentication.";
          }
        }
      } // end client-auth

      container transport-params {
        if-feature ssh-client-transport-params-config;
        uses sshcom:transport-params-grouping;
        description
          "Configurable parameters for the SSH transport layer.";
```

```
     }
   } // ssh-client-grouping
 }
```

<CODE ENDS>

3. The SSH Server Model

The SSH server model presented in this section contains one YANG grouping, for just the SSH-level configuration omitting, for instance, configuration for which ports to open to listen for connections on.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore 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 grouping defined in the ietf-ssh-server module. Please see Section 1.2 for tree diagram notation.

```
 module: ietf-ssh-server
   groupings:
   ssh-server-grouping
       +---- host-keys
       |  +---- host-key* [name]
       |     +---- name?          string
       |     +---- (host-key-type)
       |        +--:(public-key)
       |        |  +---- public-key?    -> /ks:keystore/keys/key/name
       |        +--:(certificate)
       |           +---- certificate?   leafref {sshcom:ssh-x509-certs}?
       +---- client-cert-auth {sshcom:ssh-x509-certs}?
       |  +---- trusted-ca-certs?
       |  |       -> /ks:keystore/trusted-certificates/name
       |  +---- trusted-client-certs?
       |          -> /ks:keystore/trusted-certificates/name
       +---- transport-params {ssh-server-transport-params-config}?
          +---- host-key
          |  +---- host-key-alg*   identityref
          +---- key-exchange
          |  +---- key-exchange-alg*   identityref
          +---- encryption
          |  +---- encryption-alg*   identityref
          +---- mac
          |  +---- mac-alg*   identityref
          +---- compression
             +---- compression-alg*   identityref
```

3.2. Example Usage

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

```
<!-- hypothetical example, as groupings don't have instance data -->
<ssh-server xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">

  <!-- which host-keys will this SSH server present -->
  <host-keys>
    <host-key>
      <name>deployment-specific-certificate</name>
      <certificate>ex-rsa-cert</certificate>
    </host-key>
  </host-keys>

  <!-- NOTE: password/public-key auth is NOT configured here, -->
  <!-- as it is configured in the ietf-system (RFC 7317)      -->
  <!-- module instead.                                        -->

  <!-- which client-certs will this SSH server trust -->
  <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-server>
```

3.3. YANG Model

This YANG module has a normative references to [RFC4253], [RFC6991], and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-ssh-server@2017-03-13.yang"

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

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

  import ietf-ssh-common {
    prefix sshcom;
    revision-date 2017-03-13; // stable grouping definitions
    reference
      "RFC XXXX: SSH Client and Server Models";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
```

```
    }

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

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

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


    description
     "This module defines 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 "2017-03-13" {
      description
       "Initial version";
      reference
       "RFC XXXX: SSH Client and Server Models";
    }

    // features
    feature ssh-server-transport-params-config {
      description
        "SSH transport layer parameters are configurable on an SSH
         server.";
    }

    // grouping
    grouping ssh-server-grouping {
      description
        "A reusable grouping for configuring a SSH server without
```

```
         any consideration for how underlying TCP sessions are
         established.";
       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;
             description
               "An arbitrary name for this host-key";
           }
           choice host-key-type {
             mandatory true;
             description
               "The type of host key being specified";
             leaf public-key {
               type leafref {
                 path "/ks:keystore/ks:keys/ks:key/ks:name";
               }
               description
                 "The public key is actually identified by the name of
                  its cooresponding private-key in the keystore.";
             }
             leaf certificate {
               if-feature sshcom:ssh-x509-certs;
               type leafref {
                 path "/ks:keystore/ks:keys/ks:key/ks:certificates/"
                      + "ks:certificate/ks:name";
               }
               description
                 "The name of a certificate in the keystore.";
             }
           }
         }
       }

       container client-cert-auth {
```

```
        if-feature sshcom: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 "/ks:keystore/ks:trusted-certificates/ks: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 "/ks:keystore/ks:trusted-certificates/ks: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.";
        }
      }

      container transport-params {
        if-feature ssh-server-transport-params-config;
        uses sshcom:transport-params-grouping;
        description
          "Configurable parameters for the SSH transport layer.";
      }

    } // ssh-server-grouping

  }



  <CODE ENDS>
```

4. The SSH Common Model

The SSH common model presented in this section contains identities and groupings common to both SSH clients and SSH servers. The transport-params-grouping can be used to configure the list of SSH transport algorithms permitted by the SSH client or SSH server. The

lists of algorithms are ordered such that, if multiple algorithms are permitted by the client, the algorithm that appears first in its list that is also permitted by the server is used for the SSH transport layer connection. The ability to restrict the the algorithms allowed is provided in this grouping for SSH clients and SSH servers that are capable of doing so and may serve to make SSH clients and SSH servers compliant with security policies.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive. As well, some algorithms that are REQUIRED by [RFC4253] are missing, notably "ssh-dss" and "diffie-hellman-group1-sha1" due to their weak security and there being alternatives that are widely supported.

4.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-ssh-common module. Please see Section 1.2 for tree diagram notation.

```
module: ietf-ssh-common
  groupings:
  transport-params-grouping
      +---- host-key
      |  +---- host-key-alg*   identityref
      +---- key-exchange
      |  +---- key-exchange-alg*   identityref
      +---- encryption
      |  +---- encryption-alg*   identityref
      +---- mac
      |  +---- mac-alg*   identityref
      +---- compression
         +---- compression-alg*   identityref
```

4.2. Example Usage

This section shows how it would appear if the transport-params-grouping were populated with some data.

```
  <!-- hypothetical example, as groupings don't have instance data -->
  <transport-params xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-common">

    <host-key>
      <host-key-alg>x509v3-rsa2048-sha256</host-key-alg>
      <host-key-alg>ssh-rsa</host-key-alg>
    </host-key>
    <key-exchange>
      <key-exchange-alg>
        diffie-hellman-group-exchange-sha256
      </key-exchange-alg>
    </key-exchange>
    <encryption>
      <encryption-alg>aes256-ctr</encryption-alg>
      <encryption-alg>aes192-ctr</encryption-alg>
      <encryption-alg>aes128-ctr</encryption-alg>
      <encryption-alg>aes256-cbc</encryption-alg>
      <encryption-alg>aes192-cbc</encryption-alg>
      <encryption-alg>aes128-cbc</encryption-alg>
    </encryption>
    <mac>
      <mac-alg>hmac-sha2-256</mac-alg>
      <mac-alg>hmac-sha2-512</mac-alg>
    </mac>
    <compression>
      <compression-alg>none</compression-alg>
    </compression>

  </transport-params>
```

4.3. YANG Model

This YANG module has a normative references to [RFC4344], [RFC4419], and [RFC5656].

```
  <CODE BEGINS> file "ietf-ssh-common@2017-03-13.yang"

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

    namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-common";
    prefix "sshcom";

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

    contact
```

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

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

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


    description
     "This module defines a common features, identities, and groupings
      for Secure Shell (SSH).

      Copyright (c) 2017 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 "2017-03-13" {
      description
       "Initial version";
      reference
       "RFC XXXX: SSH Client and Server Models";
    }

    // features
    feature ssh-ecc {
      description
        "Elliptic Curve Cryptography is supported for SSH.";
      reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
                   Secure Shell Transport Layer";
    }

    feature ssh-x509-certs {
      description
        "X.509v3 certificates are supported for SSH as per RFC 6187.";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
```

```
                Authentication";
    }

    feature ssh-dh-group-exchange {
      description
        "Diffie-Hellman Group Exchange is supported for SSH.";
      reference
        "RFC 4419: Diffie-Hellman Group Exchange for the
                   Secure Shell (SSH) Transport Layer Protocol";
    }

    feature ssh-ctr {
      description
        "SDCTR encryption mode is supported for SSH.";
      reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer
                   Encryption Modes";
    }

    feature ssh-sha2 {
      description
        "The SHA2 family of cryptographic hash functions is supported
         for SSH.";
      reference
        "FIPS PUB 180-4: Secure Hash Standard (SHS)";
    }

    feature ssh-zlib {
      description
        "ZLIB (LZ77) compression is supported for SSH.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    // identities
    identity public-key-alg-base {
      description
        "Base identity used to identify public key algorithms.";
    }

    identity ssh-dss {
      base public-key-alg-base;
      description
        "Digital Signature Algorithm using SHA-1 as the hashing
         algorithm.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }
```

```
    identity ssh-rsa {
      base public-key-alg-base;
      description
        "RSASSA-PKCS1-v1_5 signature scheme using SHA-1 as the hashing
         algorithm.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity ecdsa-sha2-nistp256 {
      base public-key-alg-base;
      if-feature "ssh-ecc and ssh-sha2";
      description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
         nistp256 curve and the SHA2 family of hashing algorithms.";
      reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
                   Secure Shell Transport Layer";
    }

    identity ecdsa-sha2-nistp384 {
      base public-key-alg-base;
      if-feature "ssh-ecc and ssh-sha2";
      description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
         nistp384 curve and the SHA2 family of hashing algorithms.";
      reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
                   Secure Shell Transport Layer";
    }

    identity ecdsa-sha2-nistp521 {
      base public-key-alg-base;
      if-feature "ssh-ecc and ssh-sha2";
      description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
         nistp521 curve and the SHA2 family of hashing algorithms.";
      reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
                   Secure Shell Transport Layer";
    }

    identity x509v3-ssh-rsa {
      base public-key-alg-base;
      if-feature ssh-x509-certs;
      description
        "RSASSA-PKCS1-v1_5 signature scheme using a public key stored in
         an X.509v3 certificate and using SHA-1 as the hashing
```

```
         algorithm.";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
                   Authentication";
    }

    identity x509v3-rsa2048-sha256 {
      base public-key-alg-base;
      if-feature "ssh-x509-certs and ssh-sha2";
      description
        "RSASSA-PKCS1-v1_5 signature scheme using a public key stored in
         an X.509v3 certificate and using SHA-256 as the hashing
         algorithm.  RSA keys conveyed using this format MUST have a
         modulus of at least 2048 bits.";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
         Authentication";
    }

    identity x509v3-ecdsa-sha2-nistp256 {
      base public-key-alg-base;
      if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
      description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
         nistp256 curve with a public key stored in an X.509v3
         certificate and using the SHA2 family of hashing algorithms.";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
         Authentication";
    }

    identity x509v3-ecdsa-sha2-nistp384 {
      base public-key-alg-base;
      if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
      description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
         nistp384 curve with a public key stored in an X.509v3
         certificate and using the SHA2 family of hashing algorithms.";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
         Authentication";
    }

    identity x509v3-ecdsa-sha2-nistp521 {
      base public-key-alg-base;
      if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
      description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
```

```
         nistp521 curve with a public key stored in an X.509v3
         certificate and using the SHA2 family of hashing algorithms.";
      reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
         Authentication";
    }

    identity key-exchange-alg-base {
      description
        "Base identity used to identify key exchange algorithms.";
    }

    identity diffie-hellman-group14-sha1 {
      base key-exchange-alg-base;
      description
        "Diffie-Hellman key exchange with SHA-1 as HASH and
         Oakley Group 14 (2048-bit MODP Group).";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity diffie-hellman-group-exchange-sha1 {
      base key-exchange-alg-base;
      if-feature ssh-dh-group-exchange;
      description
        "Diffie-Hellman Group and Key Exchange with SHA-1 as HASH.";
      reference
        "RFC 4419: Diffie-Hellman Group Exchange for the
                   Secure Shell (SSH) Transport Layer Protocol";
    }

    identity diffie-hellman-group-exchange-sha256 {
      base key-exchange-alg-base;
      if-feature "ssh-dh-group-exchange and ssh-sha2";
      description
        "Diffie-Hellman Group and Key Exchange with SHA-256 as HASH.";
      reference
        "RFC 4419: Diffie-Hellman Group Exchange for the
                   Secure Shell (SSH) Transport Layer Protocol";
    }

    identity ecdh-sha2-nistp256 {
      base key-exchange-alg-base;
      if-feature "ssh-ecc and ssh-sha2";
      description
        "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
         nistp256 curve and the SHA2 family of hashing algorithms.";
      reference
```

```
        "RFC 5656: Elliptic Curve Algorithm Integration in the
                   Secure Shell Transport Layer";
    }

    identity ecdh-sha2-nistp384 {
      base key-exchange-alg-base;
      if-feature "ssh-ecc and ssh-sha2";
      description
        "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
         nistp384 curve and the SHA2 family of hashing algorithms.";
      reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
                   Secure Shell Transport Layer";
    }

    identity ecdh-sha2-nistp521 {
      base key-exchange-alg-base;
      if-feature "ssh-ecc and ssh-sha2";
      description
        "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
         nistp521 curve and the SHA2 family of hashing algorithms.";
      reference
        "RFC 5656: Elliptic Curve Algorithm Integration in the
                   Secure Shell Transport Layer";
    }

    identity encryption-alg-base {
      description
        "Base identity used to identify encryption algorithms.";
    }

    identity triple-des-cbc {
      base encryption-alg-base;
      description
        "Three-key 3DES in CBC mode.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity aes128-cbc {
      base encryption-alg-base;
      description
       "AES in CBC mode, with a 128-bit key.";
      reference
       "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity aes192-cbc {
```

```
      base encryption-alg-base;
      description
        "AES in CBC mode, with a 192-bit key.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity aes256-cbc {
      base encryption-alg-base;
      description
        "AES in CBC mode, with a 256-bit key.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity aes128-ctr {
      base encryption-alg-base;
      if-feature ssh-ctr;
      description
        "AES in SDCTR mode, with 128-bit key.";
      reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
           Modes";
    }

    identity aes192-ctr {
      base encryption-alg-base;
      if-feature ssh-ctr;
      description
        "AES in SDCTR mode, with 192-bit key.";
      reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
           Modes";
    }

    identity aes256-ctr {
      base encryption-alg-base;
      if-feature ssh-ctr;
      description
        "AES in SDCTR mode, with 256-bit key.";
      reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
           Modes";
    }

    identity mac-alg-base {
      description
        "Base identity used to identify message authentication
```

```
        code (MAC) algorithms.";
    }

    identity hmac-sha1 {
      base mac-alg-base;
      description
        "HMAC-SHA1";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity hmac-sha2-256 {
      base mac-alg-base;
      if-feature "ssh-sha2";
      description
        "HMAC-SHA2-256";
      reference
        "RFC 6668: SHA-2 Data Integrity Verification for the
                   Secure Shell (SSH) Transport Layer Protocol";
    }

    identity hmac-sha2-512 {
      base mac-alg-base;
      if-feature "ssh-sha2";
      description
        "HMAC-SHA2-512";
      reference
        "RFC 6668: SHA-2 Data Integrity Verification for the
                   Secure Shell (SSH) Transport Layer Protocol";
    }

    identity compression-alg-base {
      description
        "Base identity used to identify compression algorithms.";
    }

    identity none {
      base compression-alg-base;
      description
        "No compression.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity zlib {
      base compression-alg-base;
      if-feature ssh-zlib;
      description
```

```
        "ZLIB (LZ77) compression.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    // groupings
    grouping transport-params-grouping {
      description
        "A reusable grouping for SSH transport parameters.
         For configurable parameters, a zero-element leaf-list of
         algorithms indicates the system default configuration for that
         parameter.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
      container host-key {
        description
          "Parameters regarding host key.";
        leaf-list host-key-alg {
          type identityref {
            base public-key-alg-base;
          }
          ordered-by user;
          description
            "Host key algorithms in order of descending preference.";
        }
      }
      container key-exchange {
        description
          "Parameters regarding key exchange.";
        leaf-list key-exchange-alg {
          type identityref {
            base key-exchange-alg-base;
          }
          ordered-by user;
          description
            "Key exchange algorithms in order of descending
             preference.";
        }
      }
      container encryption {
        description
          "Parameters regarding encryption.";
        leaf-list encryption-alg {
          type identityref {
            base encryption-alg-base;
          }
          ordered-by user;
          description
```

```
            "Encryption algorithms in order of descending preference.";
        }
      }
      container mac {
        description
          "Parameters regarding message authentication code (MAC).";
        leaf-list mac-alg {
          type identityref {
            base mac-alg-base;
          }
          ordered-by user;
          description
            "MAC algorithms in order of descending preference.";
        }
      }
      container compression {
        description
          "Parameters regarding compression.";
        leaf-list compression-alg {
          type identityref {
            base compression-alg-base;
          }
          ordered-by user;
          description
            "Compression algorithms in order of descending preference.";
        }
      }
    }
  }

  <CODE ENDS>
```

5. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config)

to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated 'recovery session'.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

/client-auth/password: This node is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The best reason for returning this node is to support backup/restore type workflows. This being the case, this node is marked with the NACM value 'default-deny-all'.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

6. IANA Considerations

6.1. The IETF XML Registry

This document registers three URIs in the IETF XML registry [RFC3688]. 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.

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-common
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 three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], 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

name:         ietf-ssh-common
namespace:    urn:ietf:params:xml:ns:yang:ietf-ssh-common
prefix:       sshcom
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, Juergen Schoenwaelder, Phil Shafer, Sean Turner, Michal Vasko, and Bert Wijnen.

8. References

8.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K. and G. Wu, "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), October 2016.

[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>.

[RFC4344] Bellare, M., Kohno, T., and C. Namprempre, "The Secure Shell (SSH) Transport Layer Encryption Modes", RFC 4344, DOI 10.17487/RFC4344, January 2006, <http://www.rfc-editor.org/info/rfc4344>.

[RFC4419] Friedl, M., Provos, N., and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4419, DOI 10.17487/RFC4419, March 2006, <http://www.rfc-editor.org/info/rfc4419>.

[RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer", RFC 5656, DOI 10.17487/RFC5656, December 2009, <http://www.rfc-editor.org/info/rfc5656>.

[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>.

[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>.

[RFC6668] Bider, D. and M. Baushke, "SHA-2 Data Integrity Verification for the Secure Shell (SSH) Transport Layer Protocol", RFC 6668, DOI 10.17487/RFC6668, July 2012, <http://www.rfc-editor.org/info/rfc6668>.

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

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

8.2. Informative References

[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>.

[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>.

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

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.

A.2. 00 to 01

- Renamed "keychain" to "keystore".

A.3. 01 to 02

- Removed the groupings 'listening-ssh-client-grouping' and 'listening-ssh-server-grouping'. Now modules only contain the transport-independent groupings.
- Simplified the "client-auth" part in the ietf-ssh-client module. It now inlines what it used to point to keystore for.
- Added cipher suites for various algorithms into new 'ietf-ssh-common' module.

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 Systems

EMail: garywu@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: August 27, 2017

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
February 23, 2017

Subscribing to Event Notifications
draft-ietf-netconf-subscribed-notifications-00

Abstract

This document defines capabilities and operations for subscribing to content and providing asynchronous notification message delivery on that content. 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 when using in conjunction with draft-ietf-netconf-netconf-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 August 27, 2017.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Motivation 3
1.2. Terminology 4
1.3. Solution Overview 5
2. Solution 6
2.1. Event Streams 6
2.2. Filters 6
2.3. Subscription State Model at the Publisher 6
3. Data Model Trees for Event Notifications 7
4. Dynamic Subscriptions 11
4.1. Establishing a Subscription 11
4.2. Modifying a Subscription 12
4.3. Deleting a Subscription 12
4.4. Killing a Subscription 13
5. Configured Subscriptions 13
5.1. Establishing a Configured Subscription 14
5.2. Modifying a Configured Subscription 16
5.3. Deleting a Configured Subscription 16
6. Event (Data Plane) Notifications 17
7. Subscription State Notifications 18
7.1. subscription-started 18
7.2. subscription-modified 18
7.3. subscription-terminated 19
7.4. subscription-suspended 19
7.5. subscription-resumed 19
7.6. notification-complete 19
7.7. replay-complete 19
8. Administrative Functions 20
8.1. Subscription Monitoring 20
8.2. Capability Advertisement 20
8.3. Event Stream Discovery 21
9. Data Model for Event Notifications 21
10. Considerations 41
10.1. Implementation Considerations 41
10.2. Security Considerations 41
11. Acknowledgments 42
12. References 42
12.1. Normative References 42

12.2. Informative References 43
Appendix A. Issues that are currently being worked and resolved 43
Appendix B. Changes between revisions 44
Authors' Addresses 45

1. Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service in a protocol-agnostic manner. This document defines capabilities and operations for providing asynchronous message notification delivery for notifications including those 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] (defined in [I-D.ietf-netconf-netconf-event-notif]) and Restconf [RFC8040] (defined in [I-D.ietf-netconf-restconf-notif]). The capabilities and operations defined in this document are intended to obsolete RFC 5277, along with their mapping onto NETCONF transport.

1.1. Motivation

The motivation for this work is to enable the sending of transport agnostic asynchronous notification messages driven by a YANG Subscription that are consistent with the data model (content) and security model. Predating this work was [RFC5277] which defined a limited defines a notification mechanism for for NETCONF. However, there are various [RFC5277] has limitations, many of which have been exposed in [RFC7923].

The scope of the work aims at meeting the operational needs of network subscriptions:

- Ability to dynamically or statically subscribe to event notifications available on a publisher.
- Ability to negotiate acceptable dynamic subscription parameters.
- 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 transport protocol. (In other words, the encoded notification fully describes itself.)
- Mechanism to communicate the notifications.

- Ability to replay locally logged notifications.

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].

Configured subscription: A subscription installed via a configuration interface which persists across reboots.

Dynamic subscription: A subscription agreed between subscriber and publisher created via RPC subscription state 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, 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 the Notification.

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.

NACM: NETCONF Access Control Model.

OAM: Operations, Administration, Maintenance.

Publisher: An entity responsible for streaming event notifications per the terms of a Subscriptions

Receiver: A target to which a publisher pushes event notifications. For dynamic subscriptions, the receiver and subscriber will often be the same entity.

RPC: Remote Procedure Call.

Stream (also referred to as "event stream"): A continuous ordered set of events grouped under an explicit criteria.

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

Subscription: A contract with a publisher, stipulating which information receiver(s) wishes to have pushed from the publisher without the need for further solicitation.

1.3. Solution Overview

This document describes mechanisms for subscribing and receiving event notifications from an event server publisher. This document has similarities to the capabilities orginally defined in [RFC5277]. This document extends the supported capabilties, and generalizes functionality to be protocol-agnostic.

Some enhancements over [RFC5277] include the ability to have multiple subscriptions on a single transport session, to terminate a single subscriptions without terminating the transport session, and to modify existing subscriptions.

The solution supports subscribing to event notifications using two mechanisms:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via RPC. If the publisher wants to serve this request, it will accept it, and then start pushing event notifications. If the publisher does not wish to serve it as requested, then an error response is returned. This response may include hints at subscription parameters which would have been accepted.

2. Configured subscriptions, which is an optional mechanism that enables managing subscriptions via a configuration interface so that a publisher can send event notifications to configured 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.

- 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.

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

The publisher may decide to terminate a dynamic subscription at any time. Similarly, it 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 publisher running out of resources to serve the subscription, or by internal errors on the publisher.

2. Solution

2.1. Event Streams

An event stream is a set of events available for subscription from a publisher. 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.

That said, there is one standardized event stream, this is the "NETCONF" event stream. The NETCONF event stream contains all NETCONF XML event information supported by the publisher, except for where it has been explicitly indicate that this info must be excluded from the NETCONF stream.

As events are raised by a system, they may be assigned to one or more streams. The event is distributed to receivers meeting all three critera: (1) a subscription includes the identified stream, (2) susbcription filtering allows the event to traverse, and (3) no access control rules prohibit the receiver from receiving the event.

2.2. Filters

a publisher implementation SHOULD support the ability to perform filtering of notification records per [RFC5277]. (TODO: since 5277 is to be obsoleted, we should describe the filter here.)

2.3. 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.

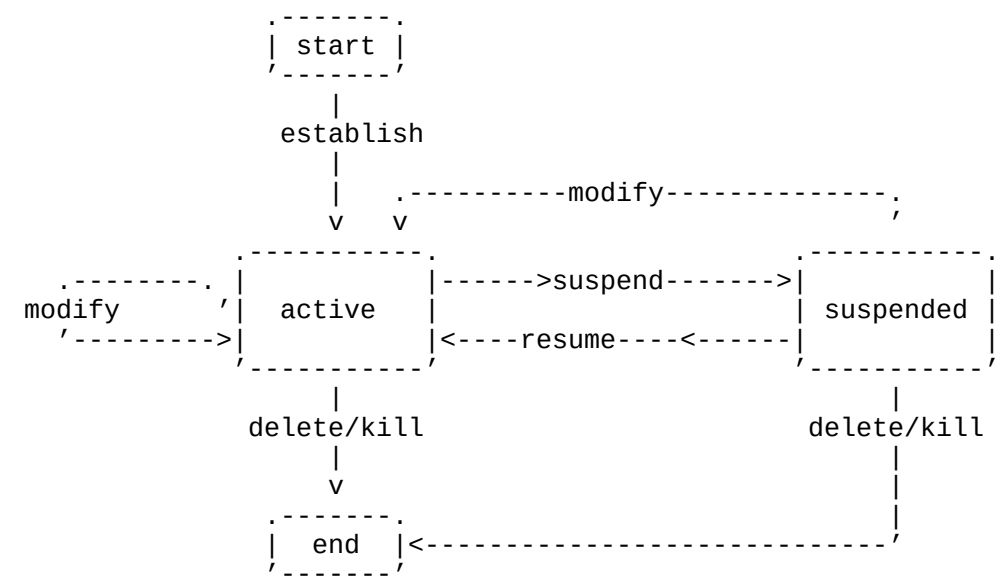


Figure 1: Subscription states at publisher

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> or <kill-subscription> will end the subscription.

3. Data Model Trees for Event Notifications

The YANG data model for event notifications is depicted in this section.

```
module: ietf-subscribed-notifications
    +--ro streams
    |  +--ro stream*   stream
    +--rw filters
    |  +--rw filter* [identifier]
    |     +--rw identifier    filter-id
    |     +--rw (filter-type)?
    |        +--:(by-reference)
```

```
|        |  +--rw filter-ref?   filter-ref
|        +--:(event-filter)
|           +--rw filter?
+--rw subscription-config {configured-subscriptions}?
|  +--rw subscription* [identifier]
|     +--rw identifier          subscription-id
|     +--rw stream?             stream
|     +--rw encoding?           encoding
|     +--rw stop-time?          yang:date-and-time
|     +--rw (filter-type)?
|     |  +--:(by-reference)
|     |  |  +--rw filter-ref?         filter-ref
|     |  +--:(event-filter)
|     |     +--rw filter?
|     +--rw receivers
|     |  +--rw receiver* [address port]
|     |     +--rw address     inet:host
|     |     +--rw port        inet:port-number
|     |     +--rw protocol?   transport-protocol
|     +--rw (notification-origin)?
|        +--:(interface-originated)
|        |  +--rw source-interface?   if:interface-ref
|        +--:(address-originated)
|           +--rw source-vrf?         string
|           +--rw source-address?     inet:ip-address-no-zone
+--ro subscriptions
   +--ro subscription* [identifier]
      +--ro identifier                 subscription-id
      +--ro configured-subscription?
      |                       empty {configured-subscriptions}?
      +--ro stream?                    stream
      +--ro encoding?                  encoding
      +--ro replay-start-time?         yang:date-and-time
      +--ro stop-time?                 yang:date-and-time
      +--ro (filter-type)?
      |  +--:(by-reference)
      |  |  +--ro filter-ref?               filter-ref
      |  +--:(event-filter)
      |     +--ro filter?
      +--ro (notification-origin)?
      |  +--:(interface-originated)
      |  |  +--ro source-interface?          if:interface-ref
      |  +--:(address-originated)
      |     +--ro source-vrf?               string
      |     +--ro source-address?           inet:ip-address-no-zone
      +--ro receivers
      |  +--ro receiver* [address]
      |     +--ro address                   inet:host
```

```
          |     +--ro port                       inet:port-number
          |     +--ro protocol?                  transport-protocol
          |     +--ro pushed-notifications?      yang:counter64
          |     +--ro excluded-notifications?    yang:counter64
          +--ro subscription-status?       subscription-status

   rpcs:
     +---x establish-subscription
     |  +---w input
     |  |  +---w stream?              stream
     |  |  +---w encoding?            encoding
     |  |  +---w replay-start-time?   yang:date-and-time
     |  |  +---w stop-time?           yang:date-and-time
     |  |  +---w (filter-type)?
     |  |     +--:(by-reference)
     |  |     |  +---w filter-ref?          filter-ref
     |  |     +--:(event-filter)
     |  |        +---w filter?
     |  +--ro output
     |     +--ro subscription-result       subscription-result
     |     +--ro (result)?
     |        +--:(no-success)
     |        |  +--ro filter-failure?           string
     |        |  +--ro replay-start-time-hint?   yang:date-and-time
     |        +--:(success)
     |           +--ro identifier               subscription-id
     +---x modify-subscription
     |  +---w input
     |  |  +---w identifier?   subscription-id
     |  |  +---w stop-time?    yang:date-and-time
     |  |  +---w (filter-type)?
     |  |     +--:(by-reference)
     |  |     |  +---w filter-ref?   filter-ref
     |  |     +--:(event-filter)
     |  |        +---w filter?
     |  +--ro output
     |     +--ro subscription-result    subscription-result
     |     +--ro (result)?
     |        +--:(no-success)
     |           +--ro filter-failure?        string
     +---x delete-subscription
     |  +---w input
     |  |  +---w identifier    subscription-id
     |  +--ro output
     |     +--ro subscription-result    subscription-result
     +---x kill-subscription
        +---w input
        |  +---w identifier    subscription-id
```

```
        +--ro output
           +--ro subscription-result    subscription-result

   notifications:
     +---n replay-complete
     |  +--ro identifier    subscription-id
     +---n notification-complete
     |  +--ro identifier    subscription-id
     +---n subscription-started
     |  +--ro identifier           subscription-id
     |  +--ro stream?              stream
     |  +--ro encoding?            encoding
     |  +--ro replay-start-time?   yang:date-and-time
     |  +--ro stop-time?           yang:date-and-time
     |  +--ro (filter-type)?
     |     +--:(by-reference)
     |     |  +--ro filter-ref?          filter-ref
     |     +--:(event-filter)
     |        +--ro filter?
     +---n subscription-resumed
     |  +--ro identifier    subscription-id
     +---n subscription-modified
     |  +--ro identifier           subscription-id
     |  +--ro stream?              stream
     |  +--ro encoding?            encoding
     |  +--ro replay-start-time?   yang:date-and-time
     |  +--ro stop-time?           yang:date-and-time
     |  +--ro (filter-type)?
     |     +--:(by-reference)
     |     |  +--ro filter-ref?          filter-ref
     |     +--:(event-filter)
     |        +--ro filter?
     +---n subscription-terminated
     |  +--ro identifier        subscription-id
     |  +--ro error-id          subscription-errors
     |  +--ro filter-failure?   string
     +---n subscription-suspended
        +--ro identifier        subscription-id
        +--ro error-id          subscription-errors
        +--ro filter-failure?   string
```

The data model is structured as follows:

- "Streams" contains a list of event streams that are supported by the publisher and that can be subscribed to.

- "Filters" contains a configurable list of filters that can be applied to a subscription. This allows users to reference an

existing filter definition as an alternative to defining a filter inline for each subscription.

- "Subscription-config" contains the configuration of configured subscriptions. The parameters of each configured subscription are a superset of the parameters of a dynamic subscription and use the same groupings. In addition, the configured subscriptions must also specify intended receivers and may specify the push source from which to send the stream of notification messages.

- "Subscriptions" contains a list of all subscriptions on a publisher, both configured and dynamic. It can be used to retrieve information about the subscriptions which an publisher is serving.

The data model also contains a number of notifications that allow a publisher to signal information about a subscription. Finally, the data model contains a number of RPC definitions that are used to manage dynamic subscriptions.

4. Dynamic Subscriptions

Dynamic subscriptions are managed via RPC.

4.1. Establishing a Subscription

The <establish-subscription> operation allows a subscriber to request the creation of a subscription via RPC.

The input parameters of the operation are:

- A filter which identifies what is being subscribed to, as well as what should be included (or not) in the pushed results.

- An optional stream which may identify or reduce the domain of events against which the subscription is applied.

- The desired encoding for the returned events. By default, updates are encoded using XML. Other encodings may be supported, such as JSON.

- An optional stop time for the subscription.

- An optional start time which indicates that this subscription is requesting a replay push of events previously generated.

If the publisher cannot satisfy the <establish-subscription> request, it sends a negative <subscription-result> element. If the subscriber

has no authorization to establish the subscription, the <subscription-result> indicates an authorization error. Optionally, the <subscription-result> may include one or more hints on alternative input parameters and value which would have resulted in an accepted subscription.

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

4.1.1. Replay Subscription

The presence of a start time indicates that this is a replay subscription. The start time must be earlier than the current time. If the start time points earlier than the maintained history of Publisher's event buffer, then the subscription must be rejected. In this case the error response to the <establish-subscription> request should include a start time supportable by the Publisher.

4.2. Modifying a Subscription

The <modify-subscription> operation permits changing the terms of an existing dynamic subscription previously established on that transport session. Subscriptions created by configuration operations cannot be modified via this RPC. Dynamic subscriptions can be modified one or multiple times. If the publisher accepts the requested modifications, it immediately starts sending events based on the new terms, completely ignoring the previous ones. If the publisher rejects the request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. The contents of a such a rejected modification may include one or more hints on alternative input parameters and value which would have resulted in a successfully modified subscription.

Dynamic subscriptions established via RPC can only be modified (or deleted) via RPC using the same transport session used to establish that subscription.

4.3. Deleting a Subscription

The <delete-subscription> operation permits canceling an existing subscription previously established on that transport session. If the publisher accepts the request, it immediately stops sending events for the subscription. If the publisher rejects the request, all subscriptions remain as prior to the request. That is, the request has no impact whatsoever.

Subscriptions established via RPC can only be deleted via RPC using the same transport session used for subscription establishment.

Configured subscriptions cannot be deleted using RPCs. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to delete configured subscriptions.

4.4. Killing a Subscription

The <kill-subscription> operation permits an operator to end any dynamic subscription. The publisher must accept the request for any dynamic subscription, and immediately stop sending events.

Configured subscriptions cannot be kill using this RPC. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to kill a configured subscription.

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.

Configured subscriptions can be modified by any configuration client with write permissions for the configuration of the subscription. Subscriptions can be modified or terminated via the configuration interface 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. If not included, push updates will go off a default interface for the device.

5.1. Establishing a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level subtree subscription-config. There are two key differences between RPC and <edit-config> RPC operations for subscription establishment. Firstly, <edit-config> operations install a subscription without question, while RPCs may support negotiation and rejection of requests. Secondly, while RPCs mandate that the subscriber establishing the subscription is the only receiver of the notifications, <edit-config> operations permit specifying receivers independent of any tracked subscriber. Immediately after a subscription is successfully established, the publisher sends to any newly active receivers a control-plane notification stating the subscription has been established (subscription-started).

Because there is no explicit association with an existing transport session, <edit-config> operations require additional parameters to indicate the receivers of the notifications and possibly the source of the notifications such as a specific egress interface.

For example at subscription establishment if NETCONF transport is being used, a 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:2.0">
            <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: Configured subscription creation via NETCONF

if the request is accepted, the publisher would reply:

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

Figure 3: Successful NETCONF configured subscription response

if the request is not accepted because the publisher cannot serve it, the publisher 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 publisher cannot serve this
            subscription due to the current workload.
        </error-message>
    </rpc-error>
</rpc-reply>
```

Figure 4: A NETCONF response for a failed configured subscription creation

5.2. 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 publisher 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, subscription-started to a specific receiver) or removed (i.e., subscription-terminated to a specific receiver.)

5.3. Deleting a Configured Subscription

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

```
DELETE /subscription-config/subscription=1922 HTTP/1.1
Host: example.com

HTTP/1.1 204 No Content
Date: Sun, 24 Jul 2016 11:23:40 GMT
Server: example-server
```

Figure 5: Deleting a configured subscription

Immediately after a subscription is successfully deleted, the publisher sends to all receivers of that subscription 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 publisher streams (asynchronously) notifications per the terms of the subscription. We refer to these as event notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the session used to establish the subscription. For configured subscriptions, event notifications are sent over the specified connections.

An event notification is sent to a receiver when something of interest occurs which is able to traverse all specified filtering and access control criteria. The event notification must include:

- a subscription-id element of type uint32 which corresponds to responsible subscription in the Publisher.
- an eventTime element which provides the time the event was generated by the event source. This event time parameter is of type dateTime and compliant to [RFC3339]. Implementations must support time zones.
- the event notification content tagged and provided by a source in the publisher.

The following is an example of a compliant event notification. This example extending the example within [RFC7950] section 7.16.3 to include the mandatory information described above:

```
<notification
       xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
        <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-id>500</subscription-id>
    <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 6: Data plane notification

While this extended [RFC7950] section 7.16 notification provides a valid method of encapsulating subscribed notifications, other transport encapsulation methods are also viable. Improvements may be achieved in some implementations in the following ways:

- transport efficiency may be gained by allowing the encapsulation and bundled push of multiple events within the same event notification.
- identifiers to designate the current and previous event notification can be used to discover duplicated and dropped notifications
- additional header types can be used to pass relevant metadata.
- a signature or hash can be included to verify the efficacy of the Publisher

This is being explored in NETMOD Notifications 2.0 [I-D.voit-notifications2].

7. Subscription State Notifications

In addition to data plane notifications, a publisher may send subscription state notifications to indicate to receivers that an event related to the subscription management has occurred.

Subscription state notifications are unlike other notifications in that they are not general-purpose notifications. They cannot be filtered out, and they are delivered only to directly impacted receiver(s) of a subscription. The definition of subscription state notifications is distinct from other notifications by making use of a YANG extension tagging them as subscription state notification.

Subscription state 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. 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.

7.2. 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.

7.3. subscription-terminated

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

Subscribers can terminate via RPC subscriptions established via a delete-subscription RPC. In such cases, no subscription-terminated notifications are sent. However if a kill-subscription RPC is sent, or some other event results in the end of a susbcription, then there must be a notification that the subscription has been ended.

7.4. subscription-suspended

This notification indicates that a publisher 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).

7.5. 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).

7.6. notification-complete

This notification is sent to indicate that a subscription, which includes a stop time, has finished passing events.

7.7. replay-complete

This notification indicates that all of the notifications prior to the current time have been sent. This includes new notifications generated since the start of the subscription. This notification must not be sent for any other reason.

If subscription contains no stop time, or has a stop time which has not been reached, then after the replay-complete notification has been sent notifications will be sent in sequence as they arise naturally within the system.

8. Administrative Functions

8.1. Subscription Monitoring

Container "subscriptions" in the YANG module below 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. Using the <get> operation with NETCONF, or subscribing to this information via [I-D.ietf-netconf-yang-push] allows the status of subscriptions to be monitored.

Each subscription is represented as a list element. 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 even if the stop time has been passed.

8.2. Capability Advertisement

Capabilities are advertised in messages sent by each peer during session establishment [RFC6241]. Publishers supporting the features in this document must advertise the capability "urn:ietf:params:netconf:capability:notification:2.0".

The mechanism defined in this document is identified by "urn:ietf:params:netconf:capability:notification:2.0". If a subscriber only supports [RFC5277] and not this specification, then they will recognize the capability "urn:ietf:params:netconf:capability:notification:1.0" and ignore the capability defined in this document.

8.3. Event Stream Discovery

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. A client can retrieve this list like any other YANG-defined data, for example using the <get> operation when using NETCONF.

9. Data Model for Event Notifications

```
<CODE BEGINS> file "ietf-subscribed-notifications.yang"
module ietf-subscribed-notifications {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications";

  prefix sn;

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

  organization "IETF";
  contact
    "WG Web:   &lt;http://tools.ietf.org/wg/netconf/&gt;
     WG List:  &lt;mailto:netconf@ietf.org&gt;

     WG Chair: Mahesh Jethanandani
               &lt;mailto:mjethanandani@gmail.com&gt;

     WG Chair: Mehmet Ersue
               &lt;mailto:mehmet.ersue@nokia.com&gt;

     Editor:   Alexander Clemm
               &lt;mailto:ludwig@clemm.org&gt;

     Editor:   Eric Voit
               &lt;mailto:evoit@cisco.com&gt;

     Editor:   Alberto Gonzalez Prieto
               &lt;mailto:albertgo@cisco.com&gt;

     Editor:   Einar Nilsen-Nygaard
```

```
               &lt;mailto:einarnn@cisco.com&gt;

     Editor:   Ambika Prasad Tripathy
               &lt;mailto:ambtripa@cisco.com&gt";


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

  revision 2017-02-23 {
    description
      "Tweaks to remove two notifications, RPC for create subscription
      refined with stream default, new grouping to eliminate some
      dymanically modifiable parameters in modifiy subscription RPC";
    reference
      "draft-ietf-netconf-subscribed-notifications-00";
  }


  /*
   * 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.";
  }

  /*
   * EXTENSIONS
   */

  extension subscription-state-notif {
    description
      "This statement applies only to notifications. It indicates that
       the notification is a subscription state notification (aka OAM
       notification). Therefore it does not participate in a regular
       event stream and does not need to be specifically subscribed
       in order to receive notifications.";
  }
```

```
  /*
   * IDENTITIES
   */

   /* Identities for streams */
   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.  This contains the same set of events in a default
       RFC-5277 NETCONF stream";
  }

  /* 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.";
  }

  /* 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 publisher 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 internal-error {
    base error;
    description
      "Error within publisher prohibits operation.";
  }

  identity suspension-timeout {
    base error;
    description
     "Termination of previously suspended subscription. The publisher
      has eliminated the subscription as it exceeded a time limit for
      suspension.";
  }

  identity stream-unavailable {
    base error;
    description
     "Stream name does not exist or is not available to the receiver.";
  }
```

```
  identity encoding-unavailable {
    base error;
    description
     "Encoding not supported";
  }

  identity replay-unsupported {
    base error;
    description
     "Replay cannot be performed for this subscription. The publisher
     does not provide the requested historic information via replay.";
  }

  identity history-unavailable {
    base error;
    description
     "Replay request too far into the past. The publisher does store
      historic information for all parts of requested subscription, but
      not back to the requested timestamp.";
  }

  identity filter-unavailable {
    base error;
    description
     "Referenced filter does not exist";
  }

  identity filter-unsupported {
    base error;
    description
     "Cannot parse syntax within the filter. Failure can be from a
      syntax error, or a syntax too complex to be processed by the
      platform. The supplemental info should include the invalid part
      of the filter.";
  }

  identity namespace-unavailable {
    base error;
    description
     "Referenced namespace doesn't exist or is unavailable
      to the receiver.";
  }

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

```
      subscriber, or an ID for acceptable subscription which has been
      statically configured.";
  }

  /* 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-errors {
    type identityref {
      base error;
    }
    description
      "The reason for the failure of an RPC request or the sending of a
       subscription suspension or termination notification";
  }

  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 notification-origin {
    type enumeration {
```

```
      enum "interface-originated" {
        description
          "Notifications will be sent from a specific interface on a
           publisher";
      }
      enum "address-originated" {
        description
          "Notifications will be sent from a specific address on a
           publisher";
      }
    }
    description
      "Specifies from where notifications will be sourced when
       being sent by the publisher.";
  }

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

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

  /*
   * GROUPINGS
   */

  grouping base-filter {
    description
      "This grouping defines the base for filters for notification
      events.";
    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 by-reference {
        description
          "Incorporate a filter that has been configured separately.";
```

```
        leaf filter-ref {
          type filter-ref;
          description
            "References an existing filter which is to be applied to
            the potential events of the subscription.";
        }
      }
      case event-filter {
        anyxml filter {
          description
           "Filter which excludes whole event-notifications. 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-policy-non-configurable {
    description
      "This grouping describes the information which can only be set
       in a dynamic subscription request via RPC.";
    leaf replay-start-time {
      type yang:date-and-time;
      description
        "Used to trigger the replay feature and indicate that the
        replay should start at the time specified.  If replay-start-time
        is not present, this is not a replay subscription and event
        pushes should start immediately.  It is never valid to
        specify start times that are later than or equal to the
        current time.";
    }
  }

  grouping subscription-policy-non-modifiable {
    description
      "This grouping describes the information in a subscription which
      should not change during the life of the subscription.";
    leaf stream {
```

```
      type stream;
      description
        "Indicates which stream of events is of interest.
         If not present, events in the default NETCONF stream
         will be sent.";
    }
    leaf encoding {
      type encoding;
      default "encode-xml";
      description
        "The type of encoding for the subscribed data.
         Default is XML";
    }
  }

  grouping subscription-policy-modifiable {
    description
      "This grouping describes all objects which may be changed
      in a subscription via an RPC.";
    leaf stop-time {
      type yang:date-and-time;
      description
        "Identifies a time after which notification events should not
        be sent.  If stop-time is not present, the notifications will
        continue until the subscription is terminated.  If
        replay-start-time exists, stop-time must for a subsequent time.
        If replay-start-time doesn't exist, stop-time must for a future
        time.";
    }
    uses base-filter;
  }

  grouping subscription-policy {
    description
      "This grouping describes information concerning a subscription.";
    uses subscription-policy-non-modifiable;
    uses subscription-policy-non-configurable;
    uses subscription-policy-modifiable;
  }

  grouping notification-origin-info {
    description
      "Defines the sender source from which notifications for a
       configured subscription are sent.";
    choice notification-origin {
      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 string;
          description
            "Network instance name for the VRF.  This could also have
            been a leafref to draft-ietf-rtgwg-ni-model, but that model
            in not complete, and may not be implemented on a box.";
        }
        leaf source-address {
          type inet:ip-address-no-zone;
          description
            "The source address for the notifications.";
        }
      }
    }
  }

  grouping receiver-info {
    description
      "Defines where and how to get notifications for a configured
      subscriptions to one or more targeted recipient.  This includes
      specifying the destination addressing as well as a transport
      protocol acceptable to the reciever.";
    container receivers {
      description
        "Set of receivers in a subscription.";
      list receiver {
        key "address port";
        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.  Each
             protocol may use the address and port information
             differently as applicable.";
        }
      }
    }
  }

  grouping error-identifier {
    description
      "A code passed back within an RPC response to describe why the RFC
       has failed, or within a state change notification to describe why
       the change has occurred.";
    leaf error-id {
      type subscription-errors;
      mandatory true;
      description
        "Identifies the subscription error condition.";
    }
  }

  grouping error-hints {
    description
      "Objects passed back within an RPC response to descibe why the RFC
       has failed, or within a state change notification to describe why
       the change has occured.";
    leaf filter-failure {
      type string;
      description
        "Information describing where and/or why a provided filter was
        unsupportable for a subscription.";
```

```
    }
  }

  grouping subscription-response-with-hints {
    description
      "Defines the output for the establish-subscription and
       modify-subscription RPCs.";
    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 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 error-hints;
      }
    }
  }


  /*
   * 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. 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 establish-subscription request.";
    input {
```

```
      uses subscription-policy;
    }
    output {
      uses subscription-response-with-hints  {
        augment "result" {
          description
            "Allows information to be passed back as part of a
             successful subscription establishment.";
          case success {
            description
              "This case is used when the subscription request was
               successful.";
            leaf identifier {
              type subscription-id;
              mandatory true;
              description
                "Identifier used for this subscription.";
            }
          }
        }
        augment "result/no-success" {
          description
            "Contains establish RPC specific objects which can be
             returned as hints for future attempts.";
          leaf replay-start-time-hint {
            type yang:date-and-time;
            description
              "If a replay has been requested, but the requested replay
                 time cannot be honored, this may provide a hint at an
                alternate time which may be supportable.";
          }
        }
      }
    }
  }

  rpc modify-subscription {
    description
      "This RPC allows a subscriber to modify a subscription that was
       previously created using establish-subscription.  If successful,
       the changed subscription remains in effect for the duration of
       the subscriber's association with the publisher, or until the
       subscription is again modified or terminated.  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 hints that would have
       a high likelihood of succeeding in a subsequent
```

```
       modify-subscription request.";
    input {
      leaf identifier {
        type subscription-id;
        description
          "Identifier to use for this subscription.";
      }
      uses subscription-policy-modifiable;
    }
    output {
      uses subscription-response-with-hints;
    }
  }

  rpc delete-subscription {
    description
      "This RPC allows a subscriber to delete a subscription that
       was previously created from by that same subscriber using the
       establish-subscription RPC.";
    input {
      leaf identifier {
        type subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be deleted.
           Only subscriptions that were created using
           establish-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.";
      }
    }
  }

  rpc kill-subscription {
    description
      "This RPC allows an operator to delete a dynamic subscription
      without restrictions on the originating subscriber or underlying
      transport session.";
    input {
      leaf identifier {
        type subscription-id;
```

```
        mandatory true;
        description
          "Identifier of the subscription that is to be deleted. Only
           subscriptions that were created using establish-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 replay-complete {
    sn:subscription-state-notif;
    description
      "This notification is sent to indicate that all of the replay
       notifications have been sent. It must not be sent for any other
       reason.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }

  notification notification-complete {
    sn:subscription-state-notif;
    description
      "This notification is sent to indicate that a subscription, has
       finished passing events.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }
```

```
  notification subscription-started {
    sn:subscription-state-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 identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses subscription-policy;
  }

  notification subscription-resumed {
    sn:subscription-state-notif;
    description
      "This notification indicates that a subscription that had
       previously been suspended has resumed. Notifications will once
       again be sent.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }

  notification subscription-modified {
    sn:subscription-state-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 identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses subscription-policy;
  }

  notification subscription-terminated {
    sn:subscription-state-notif;
    description
```

```
      "This notification indicates that a subscription has been
       terminated.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses error-identifier;
    uses error-hints;
  }

  notification subscription-suspended {
    sn:subscription-state-notif;
    description
      "This notification indicates that a suspension of the
       subscription by the publisher has occurred.  No further
       notifications will be sent until the subscription resumes.
       This notification shall only be sent to receivers of a
       subscription; it does not constitute a general-purpose
       notification.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    uses error-identifier;
    uses error-hints;
  }

  /*
   * 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 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 "identifier";
      description
        "A list of configurable filters that can be applied to
         subscriptions.";
      leaf identifier {
        type filter-id;
        description
          "An identifier to differentiate between filters.";
      }
      uses 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 "identifier";
      description
        "Content of a subscription.";
      leaf identifier {
        type subscription-id;
        description
          "Identifier to use for this subscription.";
      }
      uses subscription-policy-non-modifiable;
      uses subscription-policy-modifiable;
      uses receiver-info {
        if-feature "configured-subscriptions";
      }
      uses notification-origin-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 as well as subscriptions
       that have been established via configuration.";
    list subscription {
      key "identifier";
      config false;
      description
        "Content of a subscription.
         Subscriptions can be created using a control channel or RPC, or
         be established through configuration.";
      leaf identifier {
        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.";
      }
      uses subscription-policy;
      uses notification-origin-info {
        if-feature "configured-subscriptions";
      }
      uses receiver-info {
        augment receivers/receiver {
          description
            "include operational data on configured receivers.";
          leaf pushed-notifications {
            type yang:counter64;
            description
              "Operational data which provides the number of update
               notifications pushed to a receiver.";
            }
          leaf excluded-notifications {
            type yang:counter64;
            description
              "Operational data which provides the number of non-
               datastore update notifications explicitly removed via
               filtering so that they are not sent to a receiver.";
            }
        }
      }
```

```
      leaf subscription-status {
        type subscription-status;
        description
          "The status of the subscription.";
      }
    }
  }
}
<CODE ENDS>
```

10. Considerations

10.1. Implementation Considerations

For a deployment including both configured and dynamic subscriptions, split subscription identifiers into static and dynamic halves. That way there should not be collisions if the configured subscriptions attempt to set a subscription-id which might have already been dynamically allocated.

The <notification> elements are never sent before the transport layer, including capabilities exchange, has been established.

10.2. Security Considerations

A secure transport is highly recommended and the publisher must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of content involved. When a <get> is received that refers to the content defined in this memo, recievers should only be able to view the content for which they have sufficient privileges. <establish-subscription> 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> to which different users are able to subscribe.

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 publisher MUST NOT include any content in a notification that the user is not authorized to view.

If a malicious or buggy subscriber sends a number of <establish-subscription> requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions can be terminated by terminating the transport session. The publisher 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. Subscribers 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.

It is recommended that the NACM "very-secure" tag is placed on the <kill-subscription> RPC so that only administrators can access.

11. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Tim Jenkins, Balazs Lengyel, Shaon Chisholm, Hector Trevino, Susan Hares, Kent Watsen, Michael Scharf, and Guangying Zheng.

12. References

12.1. 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>.

[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>.

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

12.2. Informative References

[I-D.ietf-netconf-netconf-event-notif]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF support for event notifications", August 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>.

[I-D.ietf-netconf-restconf-notif]
Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", August 2016, <https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf-notif/>.

[I-D.ietf-netconf-yang-push]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", February 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

[I-D.voit-notifications2]
Voit, Eric., Clemm, Alexander., Bierman, A., and T. Jenkins, "YANG Notification Headers and Bundles", February 2017, <https://datatracker.ietf.org/draft-voit-netmod-yang-notifications2/>.

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

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

Appendix A. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

Issue #9: validate that Subscription ID will only be relevant locally to a single receiver

Issue #6: Data plane notifications and layered headers

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

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v01 5277bis - v00 subscribed notifications

- Kill subscription RPC added.
- Renamed from 5277bis to Subscribed Notifications.
- Changed the notification capabilities version from 1.1 to 2.0 as this is not RFC-5277 compatible.
- Extracted create-subscription and other elements of RFC5277.
- Error conditions added, and made specific in return codes.
- Simplified yang model structure for removal of ’basic’ grouping.
- Added a grouping for items which cannot be statically configured.
- Streams extracted if favor of more information in the filters section.
- Operational counters per receiver.
- Subscription-id and filter-id renamed to identifier
- Section for replay added. Replay-start and stop-time updated. Replay now cannot be configured.
- Control plane notification renamed to subscription state notification
- Souce address: Source-vrf changed to string, default address option added
- In yang model: ’info’ changed to ’policy’
- Scattered text clarifications

v00 - v01 of 5277bis

- YANG Model changes. New groupings for subscription info to allow restriction of what is changable via RPC. Removed notifications for adding and removing receivers of configured subscriptions.

- Expanded/renamed defintions from event server to publisher, and client to subscriber as applicable. Updated the definitions to include and expand on RFC 5277.

- Removal of redundant with other drafts

- Many other clean-ups of wording and terminology

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Systems

March 13, 2017

TLS Client and Server Models
draft-ietf-netconf-tls-client-server-02

Abstract

This document defines three YANG modules: the first defines groupings for a generic TLS client, the second defines groupings for a generic TLS server, and the third defines common identities and groupings used by both the client and the 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:

- I-D.ietf-netconf-keystore

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 I-D.ietf-netconf-keystore

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

- "2017-03-13" --> 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 3
  1.1. Terminology 3
  1.2. Tree Diagrams 4
2. The TLS Client Model 4
  2.1. Tree Diagram 4
  2.2. Example Usage 5
  2.3. YANG Model 5
3. The TLS Server Model 8
  3.1. Tree Diagram 8
  3.2. Example Usage 9
  3.3. YANG Model 9
4. The TLS Common Model 12

4.1. Tree Diagram 13
4.2. Example Usage 13
4.3. YANG Model 13
5. Security Considerations 21
6. IANA Considerations 22
6.1. The IETF XML Registry 22
6.2. The YANG Module Names Registry 22
7. Acknowledgements 22
8. References 23
8.1. Normative References 23
8.2. Informative References 24
Appendix A. Change Log 25
A.1. server-model-09 to 00 25
A.2. 00 to 01 25
A.3. 01 to 02 25
Appendix B. Open Issues 25
Authors' Addresses 25

1. Introduction

This document defines three YANG [RFC7950] modules: the first defines a grouping for a generic TLS client, the second defines a grouping for a generic TLS server, and the third defines identities and groupings common to both the client and the 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 client and server YANG modules in this document each define one grouping, which is focused on just TLS-specific configuration, and specifically avoids any transport-level configuration, such as what ports to listen-on or connect-to. This enables applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the grouping for the TLS parts it provides, while adding data nodes for the TCP-level call-home configuration.

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

The TLS client model presented in this section contains one YANG grouping, to just configure the TLS client omitting, for instance, any configuration for which IP address or port the client should connect to.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore 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 grouping defined in the ietf-tls-client module. Please see Section 1.2 for tree diagram notation.

```
   module: ietf-tls-client
     groupings:
     tls-client-grouping
         +---- server-auth
         |  +---- trusted-ca-certs?
         |  |       -> /ks:keystore/trusted-certificates/name
         |  +---- trusted-server-certs?
         |          -> /ks:keystore/trusted-certificates/name
         +---- client-auth
         |  +---- (auth-type)?
         |     +--:(certificate)
         |        +---- certificate?   leafref
         +---- hello-params {tls-client-hello-params-config}?
            +---- tls-versions
            |  +---- tls-version*   identityref
            +---- cipher-suites
               +---- cipher-suite*   identityref
```

2.2. Example Usage

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

```
<!-- hypothetical example, as groupings don't have instance data -->
<tls-client xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-client">

  <!-- which certificates will this client trust -->
  <server-auth>
    <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
    <trusted-server-certs>explicitly-trusted-client-certs</trusted-server-certs>
  </server-auth>

  <!-- how this client will authenticate itself to the server -->
  <client-auth>
    <certificate>builtin-idevid-cert</certificate>
  </client-auth>

</tls-client>
```

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [I-D.ietf-netconf-keystore].

<CODE BEGINS> file "ietf-tls-client@2017-03-13.yang"

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

     namespace "urn:ietf:params:xml:ns:yang:ietf-tls-client";
     prefix "tlsc";

     import ietf-tls-common {
       prefix tlscom;
       revision-date 2017-03-13; // stable grouping definitions
       reference
         "RFC XXXX: TLS Client and Server Models";
     }

     import ietf-keystore {
       prefix ks;
       reference
         "RFC YYYY: Keystore Model";
     }

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

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

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


     description
      "This module defines 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 "2017-03-13" {
```

```
      description
       "Initial version";
      reference
       "RFC XXXX: TLS Client and Server Models";
    }

    feature tls-client-hello-params-config {
      description
        "TLS hello message parameters are configurable on a TLS
         client.";
    }

    grouping tls-client-grouping {
      description
        "A reusable grouping for configuring a TLS client without
         any consideration for how an underlying TCP session is
         established.";

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

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

        leaf trusted-server-certs {
          type leafref {
            path "/ks:keystore/ks:trusted-certificates/ks:name";
          }
          description
            "A reference to a list of server certificates used by
             the TLS client to authenticate TLS 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 TLS server.";
```

```
          choice auth-type {
            description
              "The authentication type.";
            leaf certificate {
              type leafref {
                path "/ks:keystore/ks:keys/ks:key/ks:certificates"
                     + "/ks:certificate/ks:name";
              }
              description
                "A certificates to be used for user authentication.";
            }
          }
        }

        container hello-params {
          if-feature tls-client-hello-params-config;
          uses tlscom:hello-params-grouping;
          description
            "Configurable parameters for the TLS hello message.";
        }

      } // end tls-client-grouping

    }



    <CODE ENDS>
```

3. The TLS Server Model

The TLS server model presented in this section contains one YANG grouping, for just the TLS-level configuration omitting, for instance, configuration for which ports to open to listen for connections on.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which certificate a server should present.

3.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-tls-server module. Please see Section 1.2 for tree diagram notation.

```
   module: ietf-tls-server
     groupings:
     tls-server-grouping
         +---- certificates
         |  +---- certificate* [name]
         |     +---- name?   leafref
         +---- client-auth
         |  +---- trusted-ca-certs?
         |  |       -> /ks:keystore/trusted-certificates/name
         |  +---- trusted-client-certs?
         |          -> /ks:keystore/trusted-certificates/name
         +---- hello-params {tls-server-hello-params-config}?
            +---- tls-versions
            |  +---- tls-version*   identityref
            +---- cipher-suites
               +---- cipher-suite*   identityref
```

3.2. Example Usage

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

```
<!-- hypothetical example, groupings don't have instance data -->

<tls-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
  <certificates>
    <certificate>
      <name>tls-ec-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>
</tls-server>
```

3.3. YANG Model

This YANG module has a normative references to [RFC6991], and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-tls-server@2017-03-13.yang"

module ietf-tls-server {
```

```
yang-version 1.1;

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

import ietf-tls-common {
  prefix tlscom;
  revision-date 2017-03-13; // stable grouping definitions
  reference
    "RFC XXXX: TLS Client and Server Models";
}

import ietf-keystore {
  prefix ks;
  reference
    "RFC YYYY: Keystore Model";
}

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

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

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


description
 "This module defines 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 "2017-03-13" {
  description
```

```
        "Initial version";
      reference
       "RFC XXXX: TLS Client and Server Models";
    }

    feature tls-server-hello-params-config {
      description
        "TLS hello message parameters are configurable on a TLS
         server.";
    }

    // grouping
    grouping tls-server-grouping {
      description
        "A reusable grouping for configuring a TLS server without
         any consideration for how underlying TCP sessions are
         established.";
      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 "/ks:keystore/ks:keys/ks:key/ks:certificates/"
                   + "ks:certificate/ks:name";
            }
            description
              "The name of the certificate in the keystore.";
          }
        }
      }

      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 "/ks:keystore/ks:trusted-certificates/ks: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 "/ks:keystore/ks:trusted-certificates/ks: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.";
        }
      }

      container hello-params {
        if-feature tls-server-hello-params-config;
        uses tlscom:hello-params-grouping;
        description
          "Configurable parameters for the TLS hello message.";
      }

    } // end tls-server-grouping

  }



  <CODE ENDS>
```

4. The TLS Common Model

The TLS common model presented in this section contains identities and groupings common to both TLS clients and TLS servers. The hello-params-grouping can be used to configure the list of TLS algorithms permitted by the TLS client or TLS server. The lists of algorithms are ordered such that, if multiple algorithms are permitted by the client, the algorithm that appears first in its list that is also permitted by the server is used for the TLS transport layer connection. The ability to restrict the the algorithms allowed is

provided in this grouping for TLS clients and TLS servers that are capable of doing so and may serve to make TLS clients and TLS servers compliant with security policies.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive.

4.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-tls-common module. Please see Section 1.2 for tree diagram notation.

```
module: ietf-tls-common
  groupings:
  hello-params-grouping
      +---- tls-versions
      |  +---- tls-version*   identityref
      +---- cipher-suites
         +---- cipher-suite*   identityref
```

4.2. Example Usage

This section shows how it would appear if the transport-params-grouping were populated with some data.

```
<!-- hypothetical example, as groupings don't have instance data -->
<hello-params xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-common">

  <tls-versions>
    <tls-version>tls-1.2</tls-version>
  </tls-versions>
  <cipher-suites>
    <cipher-suite>ecdhe-rsa-with-3des-ede-cbc-sha</cipher-suite>
    <cipher-suite>dhe-rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>rsa-with-3des-ede-cbc-sha</cipher-suite>
  </cipher-suites>

</hello-params>
```

4.3. YANG Model

This YANG module has a normative references to [RFC4492], [RFC5246], [RFC5288], and [RFC5289].

```
  <CODE BEGINS> file "ietf-tls-common@2017-03-13.yang"

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

    namespace "urn:ietf:params:xml:ns:yang:ietf-tls-common";
    prefix "tlscom";

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

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

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

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


    description
     "This module defines a common features, identities, and groupings
      for Transport Layer Security (TLS).

      Copyright (c) 2017 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 "2017-03-13" {
      description
       "Initial version";
      reference
       "RFC XXXX: TLS Client and Server Models";
    }

    // features
    feature tls-ecc {
```

```
      description
        "Elliptic Curve Cryptography (ECC) is supported for TLS.";
      reference
        "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
                   for Transport Layer Security (TLS)";
    }

    feature tls-dhe {
      description
        "Ephemeral Diffie-Hellman key exchange is supported for TLS.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    feature tls-3des {
      description
        "The Triple-DES block cipher is supported for TLS.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    feature tls-gcm {
      description
        "The Galois/Counter Mode authenticated encryption mode is
         supported for TLS.";
      reference
        "RFC 5288: AES Galois Counter Mode (GCM) Cipher Suites for TLS";
    }

    feature tls-sha2 {
      description
        "The SHA2 family of cryptographic hash functions is supported
         for TLS.";
      reference
        "FIPS PUB 180-4: Secure Hash Standard (SHS)";
    }

    // identities
    identity tls-version-base {
      description
        "Base identity used to identify TLS protocol versions.";
    }

    identity tls-1.2 {
      base tls-version-base;
      description
```

```
        "TLS protocol version 1.2.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity cipher-suite-base {
      description
        "Base identity used to identify TLS cipher suites.";
    }

    identity rsa-with-aes-128-cbc-sha {
      base cipher-suite-base;
      description
        "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity rsa-with-aes-256-cbc-sha {
      base cipher-suite-base;
      description
        "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity rsa-with-aes-128-cbc-sha256 {
      base cipher-suite-base;
      if-feature tls-sha2;
      description
        "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA256.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity rsa-with-aes-256-cbc-sha256 {
      base cipher-suite-base;
      if-feature tls-sha2;
      description
        "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA256.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }
```

```
    identity dhe-rsa-with-aes-128-cbc-sha {
      base cipher-suite-base;
      if-feature tls-dhe;
      description
        "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity dhe-rsa-with-aes-256-cbc-sha {
      base cipher-suite-base;
      if-feature tls-dhe;
      description
        "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity dhe-rsa-with-aes-128-cbc-sha256 {
      base cipher-suite-base;
      if-feature "tls-dhe and tls-sha2";
      description
        "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA256.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity dhe-rsa-with-aes-256-cbc-sha256 {
      base cipher-suite-base;
      if-feature "tls-dhe and tls-sha2";
      description
        "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA256.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity ecdhe-ecdsa-with-aes-128-cbc-sha256 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256.";
      reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
```

```
    }

    identity ecdhe-ecdsa-with-aes-256-cbc-sha384 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384.";
      reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
    }

    identity ecdhe-rsa-with-aes-128-cbc-sha256 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256.";
      reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
    }

    identity ecdhe-rsa-with-aes-256-cbc-sha384 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384.";
      reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
    }

    identity ecdhe-ecdsa-with-aes-128-gcm-sha256 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-gcm and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.";
      reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
    }

    identity ecdhe-ecdsa-with-aes-256-gcm-sha384 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-gcm and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384.";
      reference
```

```
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
    }

    identity ecdhe-rsa-with-aes-128-gcm-sha256 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-gcm and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.";
      reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
    }

    identity ecdhe-rsa-with-aes-256-gcm-sha384 {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-gcm and tls-sha2";
      description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384.";
      reference
        "RFC 5289: TLS Elliptic Curve Cipher Suites with
                   SHA-256/384 and AES Galois Counter Mode (GCM)";
    }

    identity rsa-with-3des-ede-cbc-sha {
      base cipher-suite-base;
      if-feature tls-3des;
      description
        "Cipher suite TLS_RSA_WITH_3DES_EDE_CBC_SHA.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
    }

    identity ecdhe-rsa-with-3des-ede-cbc-sha {
      base cipher-suite-base;
      if-feature "tls-ecc and tls-3des";
      description
        "Cipher suite TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA.";
      reference
        "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
                   for Transport Layer Security (TLS)";
    }

    identity ecdhe-rsa-with-aes-128-cbc-sha {
      base cipher-suite-base;
      if-feature "tls-ecc";
      description
```

```
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA.";
      reference
        "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
                   for Transport Layer Security (TLS)";
    }

    identity ecdhe-rsa-with-aes-256-cbc-sha {
      base cipher-suite-base;
      if-feature "tls-ecc";
      description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA.";
      reference
        "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
                   for Transport Layer Security (TLS)";
    }

    // groupings
    grouping hello-params-grouping {
      description
        "A reusable grouping for TLS hello message parameters.  For
         configurable parameters, a zero-element leaf-list indicates the
         system default configuration for that parameter.";
      reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
                   Version 1.2";
      container tls-versions {
        description
          "Parameters regarding TLS versions.";
        leaf-list tls-version {
          type identityref {
            base tls-version-base;
          }
          description
            "Allowed TLS protocol versions.";
        }
      }
      container cipher-suites {
        description
          "Parameters regarding cipher suites.";
        leaf-list cipher-suite {
          type identityref {
            base cipher-suite-base;
          }
          ordered-by user;
          description
            "Cipher suites in order of descending preference.";
        }
      }
```

```
    }
  }
```

<CODE ENDS>

5. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated 'recovery session'.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

6. IANA Considerations

6.1. The IETF XML Registry

This document registers three URIs in the IETF XML registry [RFC3688]. 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.

URI: urn:ietf:params:xml:ns:yang:ietf-tls-common
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 three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], 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

name:         ietf-tls-common
namespace:    urn:ietf:params:xml:ns:yang:ietf-tls-common
prefix:       tlss
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, Juergen Schoenwaelder, Phil Shafer, Sean Turner, and Bert Wijnen.

8. References

8.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K. and G. Wu, "Keystore Model", draft-ietf-netconf-keystore-00 (work in progress), October 2016.

[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>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <http://www.rfc-editor.org/info/rfc4492>.

[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>.

[RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008, <http://www.rfc-editor.org/info/rfc5288>.

[RFC5289] Rescorla, E., "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)", RFC 5289, DOI 10.17487/RFC5289, August 2008, <http://www.rfc-editor.org/info/rfc5289>.

[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>.

[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>.

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

8.2. Informative References

[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>.

[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>.

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

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

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.

A.2. 00 to 01

- Renamed "keychain" to "keystore".

A.3. 01 to 02

- Removed the groupings containing transport-level configuration. Now modules contain only the transport-independent groupings.
- Filled in previously incomplete 'ietf-tls-client' module.
- Added cipher suites for various algorithms into new 'ietf-tls-common' module.

Appendix B. Open Issues

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

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com

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

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

February 28, 2017

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 1, 2017.

Copyright Notice

Table of Contents

1. Introduction 3
2. Definitions and Acronyms 5
3. Solution Overview 6
3.1. Subscription Model 6
3.2. Negotiation of Subscription Policies 7
3.3. On-Change Considerations 8
3.4. Data Encodings 9
3.5. YANG object filters 10
3.6. Push Data Stream and Transport Mapping 10
3.7. Subscription management 14
3.8. Other considerations 15
4. A YANG data model for management of datastore push subscriptions 19
4.1. Overview 19
4.2. Filters 25
4.3. Subscription configuration 26
4.4. Notifications 27
4.5. RPCs 29
5. YANG module 34
6. Security Considerations 47
7. Acknowledgments 47
8. References 47
8.1. Normative References 47
8.2. Informative References 48

Appendix A. Technologies to be considered for future iterations 49
A.1. Proxy YANG Subscription when the Subscriber and Receiver are different 49
A.2. OpState and Filters 49
A.3. Splitting push updates 50
A.4. Potential Subscription Parameters 50
Appendix B. Issues that are currently being worked and resolved 51
Appendix C. Changes between revisions 51
Authors' Addresses 52

1. Introduction

YANG [RFC7950] was originally designed for the Netconf protocol [RFC6241] which focused on configuration data. However, YANG can be used to model both configuration and operational data. It is therefore reasonable to expect YANG datastores will increasingly be used to support applications that care about about both.

For example, service assurance applications will need to be aware of any remote updates to configuration and operational objects. Rapid awareness of object changes will enable such things as validating and maintaining cross-network integrity and consistency, or monitoring state and key performance indicators of remote devices.

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

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

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

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

A more effective alternative to polling is when an application can request to be automatically updated on current relevant content of a

datastore. If such a request is accepted, interesting updates will subsequently be pushed from that datastore.

Dependence on polling-based management is typically considered an important shortcoming of applications that rely on MIBs polled using SNMP [RFC1157]. However, without a provision to support a push-based alternative, there is no reason to believe that management applications that operate on YANG datastores will be any more effective, as they would follow the same request/response pattern.

While YANG allows the definition of push notifications, such notifications generally indicate the occurrence of certain well-specified event conditions, such as the onset of an alarm condition or the occurrence of an error. A capability to subscribe to and deliver such pre-defined event notifications has been defined in [RFC5277]. In addition, configuration change notifications have been defined in [RFC6470]. These change notifications pertain only to configuration information, not to operational state, and convey the root of the subtree to which changes were applied along with the edits, but not the modified data nodes and their values. Furthermore, while delivery of updates using notifications is a viable option, some applications desire the ability to stream updates using other transports.

Accordingly, there is a need for a service that allows applications to dynamically subscribe to updates of a YANG datastore and that allows the publisher to push those updates, possibly using one of several delivery mechanisms. Additionally, support for subscriptions configured directly on the publisher are also useful when dynamic signaling is undesirable or unsupported. The requirements for such a service are documented in [RFC7923].

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

Key capabilities worth highlighting include:

- Additions to event subscription mechanisms which allow clients to subscribe to datastore updates. The subscription allows clients to specify which data they are interested in, what types of updates (e.g., create, delete, modify), and to provide filter criteria that data must meet for updates to be sent. Furthermore, subscriptions can specify a policy that directs when updates are

provided. For example, a client may request to be updated periodically in certain intervals, or whenever data changes occur.

- Format and contents of the YANG push updates themselves.

- The ability for a publisher to push back on requested subscription parameters. Because not every publisher may support every requested update policy for every piece of data, it is necessary for a publisher to be able to indicate whether or not it is capable of supporting a requested subscription, and possibly allow to hints at subscription parameters which might have succeeded.

- Subscription parameters which allow the specification of QoS extensions to address prioritization between independent streams of updates.

2. Definitions and Acronyms

Many of the terms in this document are defined in [I-D:netconf-sub-notif]. Please see that document for these definitions.

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

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

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

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

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

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

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

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

Update notification: An Event Notification including those data node update(s) to be pushed in order to meet the obligations of a single

Subscription. All included data node updates must reflect the state of a Datastore at a snapshot in time.

Update record: A representation of a data node update as a data record. An update record can be included as part of an update stream. It can also be logged for retrieval. In general, an update record will include the value/property of a data node. It may also include information about the type of data node update, i.e. whether the data node was modified/updated, or newly created, or deleted.

Update trigger: A mechanism, as specified by a Subscription Policy, that determines when a data node update is to be communicated. (e.g., a change trigger, invoked when the value of a data node changes or a data node is created or deleted, or a time trigger, invoked after the laps of a periodic time interval.)

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

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

3. Solution Overview

This document specifies a solution for a push update subscription service. This solution supports the dynamic as well as configured subscriptions to information updates from YANG datastores. A subscription might target exposed operational and/or configuration YANG objects on a device. YANG objects are subsequently pushed from the publisher to the receiver per the terms of the subscription.

3.1. Subscription Model

YANG-push subscriptions are defined using a data model that is itself defined in YANG. This model augments the event subscription model defined in [I-D:netconf-sub-notif] and introduces new capabilities that allow subscribers to specify what to include in an update notification and what triggers such an update notification.

- Enhancements to filters. Specifically the filter must at least identify at least one targeted yang data node/subtree. The filter may also define additional yang nodes/subtrees to include or exclude. The publisher must only send to the receiver those data node updates that can traverse applied filter. Filters can be specified "inline" as part of the subscription, or can be configured separately and referenced by a subscription in order to facilitate reuse of complex filters.

- o A subscription policy definition regarding the update trigger when to send new update notifications.
  - * For periodic subscriptions, the trigger is defined by two parameters that defines the interval with which updates are to be pushed. These parameters are the period/interval of reporting duration, and an anchor time which can be used to calculate at which times updates needs to be assembled and sent.
  - * For on-change subscriptions, the trigger occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that can be guided by additional parameters. Please refer also to Section 3.3.
    - + One parameter specifiing the dampening period. This period is the interval which must pass before a successive update notification for the same Subscription is sent. Note that the dampening period applies to the set of all data nodes within a single subscription. This means that on the first change of an object, an update notification containing that object is sent either immediately or at the end of a dampening period already in effect.
    - + Another parameter allowing the restriction of the types of changes for which updates are sent (changes to object values, object creation or deletion events).
    - + A third parameter specifing whether or not a complete push-update with all the subscribed data should be sent at the beginning of a subscription. Such a push provides the receiver the current state, and establish the frame of reference for subsequent updates.
- o Anydata encoding for the contents of periodic and on-change push updates.

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

3.2. Negotiation of Subscription Policies

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

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

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

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

3.3. On-Change Considerations

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

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

When an on-change subscription is requested for a datastream with a given subtree filter, where not all objects support on-change update triggers, only the objects supporting on-change will be provided. For more on how objects are so marked, see Section 3.8.5

Any updates for an on-change subscription will include only supported objects for which a change was detected and which met the filtering criteria. To avoid flooding receivers with repeated updates for fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update for a given object is sent, no other updates for this particular subscription are sent until the end of the dampening

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

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

3.4. Data Encodings

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

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

3.4.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been simply retrieved using a get operation and is encoded in the same way. XML encoding rules for data nodes are defined in [RFC7950]. JSON encoding rules are defined in [RFC7951]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

3.4.2. On-Change Subscriptions

In an on-change subscription, updates need to indicate not only values of changed data nodes but also the types of changes that occurred since the last update, such as whether data nodes were newly created since the last update or whether they were merely modified, as well as which data nodes were deleted.

Encoding rules for data in on-change updates correspond to how data would be encoded in a YANG-patch operation as specified in [RFC8072]. The "YANG-patch" would in this case be applied to the earlier state reported by the preceding update, to result in the now-current state of YANG data. Of course, contrary to a YANG-patch operation, the data is sent from the publisher to the receiver and is not restricted to configuration data.

3.5. YANG object filters

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

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

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

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

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

3.6. Push Data Stream and Transport Mapping

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

An applicable mechanism is that of a notification. There are however some specifics that need to be considered. Contrary to other notifications that are associated with alarms and unexpected event occurrences, update notifications are solicited, i.e. tied to a particular subscription which triggered the notification.

A push update notification contains several parameters:

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

- Data nodes containing a representation of the datastore subtree(s) containing the updates. In all cases, the subtree(s) are filtered per access control rules to contain only data that the subscriber is authorized to see. For on-change subscriptions, the subtree may only contain the data nodes which have changed since the start of the previous dampening interval.

This document introduces two generic notifications: "push-update" and "push-change-update". Those notifications may be encapsulated on a

transport (e.g., NETCONF or HTTP) to carry data records with updates of datastore contents as specified by a subscription. It is possible also map notifications to other transports and encodings and use the same subscription model; however, the definition of such mappings is outside the scope of this document.

A push-update notification defines a complete update of the datastore per the terms of a subscription. This type of notification is used for continuous updates of periodic subscriptions. A push-update notification can also used be for the on-change subscriptions in two cases. First it will be used as the initial push-update if there is a need to synchronize the receiver at the start of a new subscription. It also may be sent if the publisher later chooses to resynch a previously synched on-change subscription. The push-update record contains a data snippet that contains an instantiated subtree with the subscribed contents. The content of the update notification is equivalent to the contents that would be obtained had the same data been explicitly retrieved using e.g., a Netconf "get"-operation, with the same filters applied.

The contents of the push-update notification conceptually represents the union of all data nodes in the yang modules supported by the publisher. However, in a YANG data model, it is not practical to model the precise data contained in the updates as part of the notification. To capture this data, a single parameter that can encapsulate the full set of subscribed datastore contents is used, not parameters that represent data nodes one at a time.

A push-change-update notification is the most common type of update for on-change subscriptions. The update record in this case contains a data snippet that indicates the full set of changes that data nodes have undergone since the last notification of YANG objects. In other words, this indicates which data nodes have been created, deleted, or have had changes to their values. The format of the data snippet follows YANG-patch [RFC8072], i.e. the same format that would be used with a YANG-patch operation to apply changes to a data tree, indicating the creates, deletes, and modifications of data nodes. Please note that as the update can include a mix of configuration and operational data

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

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-update
       xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>1011</subscription-id>
     <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
     <datastore-contents-xml>
        <foo>
           <bar>some_string</bar>
        </foo>
     </datastore-contents-xml>
   </push-update>
</notification>
```

Figure 1: Push example

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

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-change-update xmlns=
       "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>89</subscription-id>
     <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
     <datastore-changes-xml>
       <alpha xmlns="http://example.com/sample-data/1.0" >
         <beta>1500</beta>
       </alpha>
     </datastore-changes-xml>
   </push-change-update>
</notification>
```

Figure 2: Push example for on change

The equivalent update when requesting json encoding:

```
<notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
   <eventTime>2015-03-09T19:14:56Z</eventTime>
   <push-change-update xmlns=
       "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
     <subscription-id>89</subscription-id>
     <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
     <datastore-changes-json>
       {
        "ietf-yang-patch:yang-patch": {
        "patch-id": [
          null
        ],
        "edit": [
          {
              "edit-id": "edit1",
              "operation": "merge",
              "target": "/alpha/beta",
              "value": {
                  "beta": 1500
              }
          }
        ]
       }
     }
     </datastore-changes-json>
   </push-change-update>
</notification>
```

Figure 3: Push example for on change with JSON

When the beta leaf is deleted, the publisher may send

```
        <notification
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
           <eventTime>2015-03-09T19:14:56Z</eventTime>
           <push-change-update xmlns=
               "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
             <subscription-id>89</subscription-id>
             <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
             <datastore-changes-xml>
               <alpha xmlns="http://example.com/sample-data/1.0" >
                 <beta urn:ietf:params:xml:ns:netconf:base:1.0:
                     operation="delete"/>
               </alpha>
             </datastore-changes-xml>
           </push-change-update>
        </notification>
```

Figure 4: 2nd push example for on change update

3.7. Subscription management

A [I-D:netconf-sub-notif] subscription needs enhancment to support YANG Push subscription negotiation. Specifically, these enhancements are needed to signal to the subscriber why an attempt has failed.

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

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

For instance, for the following request:

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

Figure 5: Establish-Subscription example

the publisher might return:

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

Figure 6: Error response example

3.8. Other considerations

3.8.1. Authorization

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

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

One of these clarifications is that a subscription may only be established if the receiver has read access to every data node specifically named within the subscription filter.

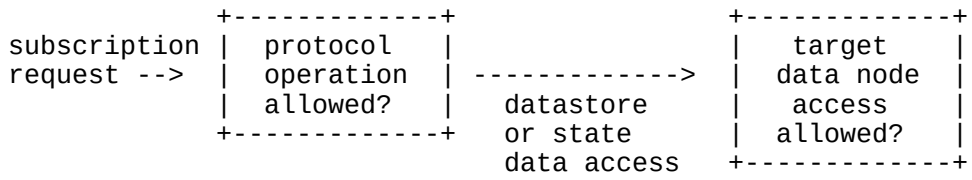


Figure 7: Access control for subscription

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

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

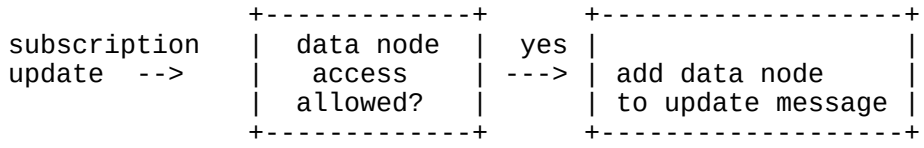


Figure 8: Access control for push updates

If there are read access control changes applied under the data node named/targeted within a filter, no notifications indicating the fact that this has occurred should be provided.

3.8.2. Robustness and reliability considerations

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

Update notifications will typically traverse a secure and reliable transport. Notifications will not be reordered, and will also contain a time stamp. Despite these protections for on-change, it is possible that complete update notifications get lost. For this reason, update sequence numbers for push-change-updates may be included in a subscription so that an application can determine if an update has been lost.

At the same time, it is conceivable that under certain circumstances, a publisher will recognize that it is unable to include within an update notification the full set of objects desired per the terms of a subscription. In this case, the publisher must take one or more of the following actions.

- A publisher must set the updates-not-sent flag on any update notification which is known to be missing information.
- It may choose to suspend and resume a subscription as per [I-D:netconf-sub-notif].
- When resuming an on-change subscription, the publisher should generate a complete patch from the previous update notification. If this is not possible and the synch-on-start option is configured, then the full datastore contents may be sent instead (effectively replacing the previous contents). If neither of these are possible, then an updates-not-sent flag must be included on the next push-change-update.

3.8.3. Update size and fragmentation considerations

Depending on the subscription, the volume of updates can become quite large. Additionally, based on the platform, it is possible that push-updates for a single subscription are best sent independently from different line-cards. Therefore, it may not always be practical to send the entire update in a single chunk. Implementations of push-update MAY therefore choose, at their discretion, to "chunk" updates and break them out into several push-update notifications. In this case the updates-not-sent flag will indicate that no single push-update is complete. Push-change-updates may also be chunked as long as none of the changed objects in the separate pushes are state-entangled.

3.8.4. Implementation considerations

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

Even periodic retrieval and push of operational counters may consume considerable system resources. In addition the on-change push of small amounts of configuration data may, depending on the implementation, require invocation of APIs, possibly on an object-by-object basis, possibly involving additional internal interrupts, etc.

For those reasons, it is important for an implementation to understand what subscriptions it can or cannot support. It is far preferable to decline a subscription request then to accept such a request when it cannot be met.

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

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

3.8.5. Identifying on-change notifiable YANG objects

In some cases, a publisher supporting on-change notifications may not be able to push updates for some object types on-change. Reasons for this might be that the value of the data node changes frequently (e.g., a received-octets-counter), that small object changes are frequent and meaningless (e.g., a temperature gauge changing 0.1 degrees), or that the implementation is not capable of on-change notification of an object type.

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

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

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

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

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

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

   argument value;
 }
```

Figure 9: Notifiable Extension

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

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

Figure 10: Deviation Example

4. A YANG data model for management of datastore push subscriptions

4.1. Overview

The YANG data model for datastore push subscriptions is depicted in the following figure. Following Yang tree convention in the depiction, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parantheses with a name in the middle enclose choice and case nodes. A "+" at the end of a line indicates that the line is to be concatenated with the subsequent line. New YANG tree notation is the i] which indicates that the node in that line has been brought in / imported from another model, and an (a) which indicates this is the specific imported node augmented. In the figure below, all have been imported from 5277bis. The model consists mostly of augmentations to RPCs and notifications defined in

the data model for subscriptions for event notifications of [I-D:netconf-sub-notif].

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

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

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

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

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

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

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

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

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

Figure 11: Model structure

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

4.2. Filters

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

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

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

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

4.3. Subscription configuration

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

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

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

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

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

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

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

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

* a "dampening-period" specifies the interval that must pass before a successive update for the subscription is sent. The first time a change is detected, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed for this subscription has passed.

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

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

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

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

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

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

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

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

4.4. Notifications

4.4.1. Monitoring and OAM Notifications

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

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

4.4.2. Update Notifications

The data model introduces two YANG notifications for the actual updates themselves.

Notification "push-update" is used to send a complete snapshot of the data that has been subscribed to, with all YANG object filters applied. The notification is used for periodic subscription updates in a periodic subscription.

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

The format and syntax of the contained update notification data corresponds to the format and syntax of data that would be returned in a corresponding get operation with the same filter parameters applied.

Notification "push-change-update" is used to send data updates for changes that have occurred in the subscribed data. This notification is used only in conjunction with on-change subscriptions.

The data updates are encoded analogous to the syntax of a corresponding yang-patch operation. It corresponds to the data that would be contained in a yang-patch operation applied to the YANG

datastore at the previous update, to result in the current state (and applying it also to operational data).

If the application detects a discontinuity in the updates it is pushing, the notification can include a flag "updates-not-sent". This is a flag which indicates that not all changes which have occurred since the last update are actually included with this update. In other words, the publisher has failed to fulfill its full subscription obligations, for example in cases where it was not able to keep up with a change burst. To facilitate synchronization, a publisher MAY subsequently send a push-update containing a full snapshot of subscribed data.

4.5. RPCs

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

4.5.1. Establish-subscription RPC

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

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

Figure 12: Establish-subscription RPC

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

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

Figure 13: Establish-subscription positive RPC response

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

When the requester is not authorized to read the requested data node, the returned information indicates the node is unavailable. For instance, if the above request was unauthorized to read node "ex:foo" the publisher may return:

```
<rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <subscription-result
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       subtree-unavailable
    </subscription-result>
    <filter-failure
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       /ex:foo
    </filter-failure>
</rpc-reply>
```

Figure 14: Establish-subscription access denied response

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

For example, for the following request:

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

Figure 15: Establish-subscription request example 2

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

```
<rpc-reply message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <subscription-result
          xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
       period-unsupported
    </subscription-result>
    <period-hint>100</period-hint>
</rpc-reply>
```

Figure 16: Establish-subscription error response example 2

4.5.2. Modify-subscription RPC

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

```
<netconf:rpc message-id="102"
   xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
   <modify-subscription
        xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      <stream>push-update</stream>
      <subscription-id>
         1011
      </subscription-id>
      <period>250</period>
   </modify-subscription>
</netconf:rpc>
```

Figure 17: Modify subscription request

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

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

Figure 18: Modify subscription response

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

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

4.5.3. Delete-subscription RPC

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

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

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

Figure 19: Delete subscription

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

4.5.4. YANG Module Synchronization

In order to fully support datastore replication, the receiver needs to know the YANG module library that is in use by server that is being replicated. The YANG 1.0 module library information is sent by a NETCONF server in the NETCONF 'hello' message. For YANG 1.1 modules and all modules used with the RESTCONF [RFC8040] protocol, this information is provided by the YANG Library module (ietf-yang-library.yang from [RFC7895]. The YANG library information is important for the receiver to reproduce the set of object definitions used by the replicated datastore.

The YANG library includes a module list with the name, revision, enabled features, and applied deviations for each YANG module implemented by the publisher. The receiver is expected to know the YANG library information before starting a subscription. The "/modules-state/module-set-id" leaf in the "ietf-yang-library" module can be used to cache the YANG library information.

The set of modules, revisions, features, and deviations can change at run-time (if supported by the server implementation). In this case, the receiver needs to be informed of module changes before data nodes from changed modules can be processed correctly. The YANG library provides a simple "yang-library-change" notification that informs the client that the library has changed somehow. The receiver then needs

to re-read the entire YANG library data for the replicated server in order to detect the specific YANG library changes. The "ietf-netconf-notifications" module defined in [RFC6470] contains a "netconf-capability-change" notification that can identify specific module changes. For example, the module URI capability of a newly loaded module will be listed in the "added-capability" leaf-list, and the module URI capability of an removed module will be listed in the "deleted-capability" leaf-list.

5. YANG module

```
<CODE BEGINS> file "ietf-yang-push@2017-02-08.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

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

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

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

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

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

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

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

     Editor:   Ambika Prasad Tripathy
```

```
               <mailto:ambtripa@cisco.com>

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

     Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>

     Editor:   Balazs Lengyel
               <mailto:balazs.lengyel@ericsson.com>";

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

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

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

 /*
  * IDENTITIES
  */

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

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

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

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

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

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

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

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

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

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

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

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

  /* Additional transport option */
  identity http2 {
    base sn:transport;
    description
      "HTTP2 notifications as a transport";
  }

  /*
   * TYPE DEFINITIONS
   */

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

  typedef change-type {
    type enumeration {
      enum "create" {
        description
          "A new data node was created";
      }
      enum "delete" {
        description
          "A data node was deleted";
      }
      enum "modify" {
        description
          "The value of a data node has changed";
      }
    }
    description
      "Specifies different types of changes that may occur to a
       datastore.";
  }

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

```
       datastore tree.  The filters define which updates are of
       interest in a push update subscription.  Mixing and matching
       of multiple filters does not occur at the level of this
       grouping. When a push-update subscription is created, the
       filter can be a regular subscription filter, or one of the
       additional filters that are defined in this grouping.";
    choice update-filter {
      description
        "Define filters regarding which data nodes to include
         in push updates";
      case subtree {
        description
          "Subtree filter.";
        anyxml subtree-filter {
          description
            "Subtree-filter used to specify the data nodes targeted
             for subscription within a subtree, or subtrees, of a
             conceptual YANG datastore.  Objects matching the filter
             criteria will traverse the filter. The syntax follows
             the subtree filter syntax specified in RFC 6241.";
          reference "RFC 6241 section 6";
        }
      }
      case xpath {
        description
          "XPath filter";
        leaf xpath-filter {
          type yang:xpath1.0;
          description
            "Xpath defining the data items of interest.";
        }
      }
    }
  }

  grouping update-policy-modifiable {
    description
      "This grouping describes the datastore specific subscription
       conditions that can be changed during the lifetime of the
       subscription.";
    choice update-trigger {
      description
        "Defines necessary conditions for sending an event to
         the subscriber.";
      case periodic {
        description
          "The agent is requested to notify periodically the current
           values of the datastore as defined by the filter.";
```

```
        leaf period {
          type yang:timeticks;
          mandatory true;
          description
            "Duration of time which should occur between periodic
             push updates.  Where the anchor of a start-time is
             available, the push will include the objects and their
             values which exist at an exact multiple of timeticks
             aligning to this start-time anchor.";
        }
        leaf anchor-time {
          type yang:date-and-time;
          description
            "Designates a timestamp from which the series of periodic
             push updates are computed. The next update will take place
             at the next period interval from the anchor time.  For
             example, for an anchor time at the top of a minute and a
             period interval of a minute, the next update will be sent
             at the top of the next minute.";
        }
      }
      case on-change {
        if-feature "on-change";
        description
          "The agent is requested to notify changes in values in the
           datastore subset as defined by a filter.";
        leaf dampening-period {
          type yang:timeticks;
          mandatory true;
          description
            "Minimum amount of time that needs to have passed since the
             last time an update was provided for the subscription.";
        }
      }
    }
  }

  grouping update-policy {
    description
      "This grouping describes the datastore specific subscription
       conditions of a subscription.";
    uses update-policy-modifiable {
      augment "update-trigger/on-change" {
        description
          "Includes objects not modifiable once subscription is
           established.";
        leaf no-synch-on-start {
          type empty;
```

```
          description
            "This leaf acts as a flag that determines behavior at the
             start of the subscription.  When present, synchronization
             of state at the beginning of the subscription is outside
             the scope of the subscription. Only updates about changes
             that are observed from the start time, i.e. only push-
             change-update notifications are sent. When absent (default
             behavior), in order to facilitate a receiver's
             synchronization, a full update is sent when the
             subscription starts using a push-update notification, just
             like in the case of a periodic subscription.  After that,
             push-change-update notifications only are sent unless the
             Publisher chooses to resynch the subscription again.";
        }
        leaf-list excluded-change {
          type change-type;
          description
            "Use to restrict which changes trigger an update.
             For example, if modify is excluded, only creation and
             deletion of objects is reported.";
        }
      }
    }
  }

  grouping update-qos {
    description
      "This grouping describes Quality of Service information
       concerning a subscription.  This information is passed to lower
       layers for transport prioritization and treatment";
    leaf dscp {
      type inet:dscp;
      default "0";
      description
        "The push update's IP packet transport priority. This is made
         visible across network hops to receiver. The transport
         priority is shared for all receivers of a given
         subscription.";
    }
    leaf weighting {
      type uint8 {
         range "0 .. 255";
      }
      description
        "Relative weighting for a subscription. Allows an underlying
         transport layer perform informed load balance allocations
         between various subscriptions";
      reference
```

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

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

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

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

```
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
  augment "/sn:modify-subscription/sn:output/"+
    "sn:result/sn:no-success" {
    description
      "Add push datastore error info and hints to RPC output.";
    uses update-error-hints;
  }

  notification push-update {
    description
      "This notification contains a push update, containing data
       subscribed to via a subscription. This notification is sent for
       periodic updates, for a periodic subscription.  It can also be
       used for synchronization updates of an on-change subscription.
       This notification shall only be sent to receivers of a
       subscription; it does not constitute a general-purpose
       notification.";
    leaf subscription-id {
      type sn:subscription-id;
      mandatory true;
      description
        "This references the subscription because of which the
         notification is sent.";
    }
    leaf time-of-update {
      type yang:date-and-time;
      description
        "This leaf contains the time of the update.";
    }
    leaf updates-not-sent {
      type empty;
      description
        "This is a flag which indicates that not all data nodes
         subscribed to are included with this update. In other words,
         the publisher has failed to fulfill its full subscription
         obligations. This may lead to intermittent loss of
         synchronization of data at the client.  Synchronization at the
         client can occur when the next push-update is received.";
    }
    anydata datastore-contents {
      description
        "This contains the updated data.  It constitutes a snapshot
         at the time-of-update of the set of data that has been
         subscribed to.  The format and syntax of the data
```

```
          corresponds to the format and syntax of data that would be
          returned in a corresponding get operation with the same
          filter parameters applied.";
    }
  }
  notification push-change-update {
    if-feature "on-change";
    description
      "This notification contains an on-change push update. This
       notification shall only be sent to the receivers of a
       subscription; it does not constitute a general-purpose
       notification.";
    leaf subscription-id {
      type sn:subscription-id;
      mandatory true;
      description
        "This references the subscription because of which the
         notification is sent.";
    }
    leaf time-of-update {
      type yang:date-and-time;
      description
        "This leaf contains the time of the update, i.e. the time at
         which the change was observed.";
    }
    leaf updates-not-sent {
      type empty;
      description
        "This is a flag which indicates that not all changes which
         have occurred since the last update are included with this
         update.  In other words, the publisher has failed to
         fulfill its full subscription obligations, for example in
         cases where it was not able to keep up with a change burst.
         To facilitate synchronization, a publisher MAY subsequently
         send a push-update containing a full snapshot of subscribed
         data. Such a push-update might also be triggered by a
         subscriber requesting an on-demand synchronization.";
    }
    anydata datastore-changes {
      description
        "This contains datastore contents that has changed since the
         previous update, per the terms of the subscription.  Changes
         are encoded analogous to the syntax of a corresponding yang-
         patch operation, i.e. a yang-patch operation applied to the
         YANG datastore implied by the previous update to result in the
         current state (and assuming yang-patch could also be applied to
         operational data).";
    }
```

```
  }
  augment "/sn:subscription-started" {
    description
      "This augmentation adds push subscription parameters to the
       notification that a subscription has started and data updates are
       beginning to be sent. This notification shall only be sent to
       receivers of a subscription; it does not constitute a general-
       purpose notification.";
    uses update-policy;
    uses update-qos;
  }
  augment "/sn:subscription-started/sn:filter-type" {
    description
      "This augmentation allows to include additional update filters
       options to be included as part of the notification that a
       subscription has started.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
  augment "/sn:subscription-modified" {
    description
      "This augmentation adds push subscription parameters to the
       notification that a subscription has been modified. This
       notification shall only be sent to receivers of a subscription;
       it does not constitute a general-purpose notification.";
    uses update-policy;
    uses update-qos;
  }
  augment "/sn:subscription-modified/sn:filter-type" {
    description
      "This augmentation allows to include additional update
       filters options to be included as part of the notification
       that a subscription has been modified.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
  augment "/sn:filters/sn:filter/"+
          "sn:filter-type" {
    description
      "This container adds additional update filter options to the list
       of configurable filters that can be applied to subscriptions.
       This facilitates the reuse of complex filters once defined.";
```

```
    case update-filter {
      uses update-filter;
    }
  }
  augment "/sn:subscription-config/sn:subscription" {
    description
      "Contains the list of subscriptions that are configured,
       as opposed to established via RPC or other means.";
    uses update-policy;
    uses update-qos;
  }
  augment "/sn:subscription-config/sn:subscription/"+
          "sn:filter-type" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      uses update-filter;
    }
  }
  augment "/sn:subscriptions/sn:subscription" {
    description
      "Contains the list of currently active subscriptions,
       i.e. subscriptions that are currently in effect,
       used for subscription management and monitoring purposes.
       This includes subscriptions that have been setup via RPC
       primitives, e.g. establish-subscription, delete-subscription,
       and modify-subscription, as well as subscriptions that
       have been established via configuration.";
    uses update-policy;
    uses update-qos;
  }
  augment "/sn:subscriptions/sn:subscription/"+
          "sn:filter-type" {
    description
      "Add push filters to selection of filter types.";
    case update-filter {
      description
        "Additional filter options for push subscription.";
      uses update-filter;
    }
  }
}

<CODE ENDS>
```

6. Security Considerations

Subscriptions could be used to attempt to overload publishers of YANG datastores. For this reason, it is important that the publisher has the ability to decline a subscription request if it would deplete its resources. In addition, a publisher needs to be able to suspend an existing subscription when needed. When this occur, the subscription status is updated accordingly and the receivers are notified. Likewise, requests for subscriptions need to be properly authorized.

A subscription could be used to retrieve data in subtrees that a receiver has no authorized access to. Therefore it is important that data pushed based on subscriptions is authorized in the same way that regular data retrieval operations are. Data being pushed to a receiver needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model applies.

A subscription could be configured on another receiver's behalf, with the goal of flooding that receiver with updates. One or more publishers could be used to overwhelm a receiver which doesn't even support subscriptions. Receivers which do not want pushed data need only terminate or refuse any transport sessions from the publisher. In addition, the Netconf Authorization Control Model SHOULD be used to control and restrict authorization of subscription configuration.

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

7. Acknowledgments

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

8. References

8.1. Normative References

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

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

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

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

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

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

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

8.2. Informative References

[RFC1157] Case, J., "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, June 2016.

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

Appendix A. Technologies to be considered for future iterations

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

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

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

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

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

A.2. OpState and Filters

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

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

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

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

A.3. Splitting push updates

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

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

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

A.4. Potential Subscription Parameters

A possible is the introduction of an additional parameter "changes-only" for periodic subscription. Including this flag would results in sending at the end of each period an update containing only changes since the last update (i.e. a change-update as in the case of an on-change subscription), not a full snapshot of the subscribed

information. Such an option might be interesting in case of data that is largely static and bandwidth-constrained environments.

Appendix B. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

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

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v04 to v05

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

v03 to v04

- Updates-not-sent flag added
- Not notifiable extension added
- Dampening period is for whole subscription, not single objects

- Moved start/stop into rfc5277bis
- Client and Server changed to subscriber, publisher, and receiver
- Anchor time for periodic
- Message format for synchronization (i.e. synch-on-start)
- Material moved into 5277bis
- QoS parameters supported, by not allowed to be modified by RPC
- Text updates throughout

Authors' Addresses

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Balazs Lengyel
Ericsson

Email: balazs.lengyel@ericsson.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

K. Watsen
Juniper Networks
M. Abrahamsson
T-Systems
March 13, 2017

Zero Touch Provisioning for NETCONF or RESTCONF based Management
draft-ietf-netconf-zerotouch-13

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:

- I-D.ieft-netconf-netconf-client-server
- I-D.ietf-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:

- "2017-03-13" --> the publication date of this draft

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

- Appendix A. Change Log

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 4
1.1. Use Cases 4
1.2. Terminology 5
1.3. Requirements Language 6
1.4. Tree Diagram Notation 6
2. Guiding Principles 7
2.1. Trust Anchors 7
2.2. Conveying Trust 7
2.3. Conveying Ownership 8
3. Types of Zero Touch Information 8
3.1. Redirect Information 8
3.2. Bootstrap Information 9
4. Artifacts 10
4.1. Zero Touch Information 10

4.2. Owner Certificate 11
4.3. Ownership Voucher 12
5. Artifact Groupings 12
5.1. Unsigned Information 12
5.2. Signed Information (without Revocations) 13
5.3. Signed Information (with Revocations) 13
6. Sources of Bootstrapping Data 14
6.1. Removable Storage 14
6.2. DNS Server 15
6.3. DHCP Server 16
6.4. Bootstrap Server 17
7. Workflow Overview 18
7.1. Onboarding and Ordering Devices 19
7.2. Owner Stages the Network for Bootstrap 21
7.3. Device Powers On 23
8. Device Details 25
8.1. Factory Default State 25
8.2. Boot Sequence 26
8.3. Processing a Source of Bootstrapping Data 27
8.4. Validating Signed Data 28
8.5. Processing Redirect Information 29
8.6. Processing Bootstrap Information 30
9. The Zero Touch Information Artifact 31
9.1. Tree Diagram 31
9.2. Example Usage 31
9.3. YANG Module 34
10. The Zero Touch Bootstrap Server API 39
10.1. Tree Diagram 39
10.2. Example Usage 40
10.3. YANG Module 43
11. Security Considerations 51
11.1. Immutable storage for trust anchors 51
11.2. Clock Sensitivity 51
11.3. Blindly authenticating a bootstrap server 51
11.4. Entropy loss over time 52
11.5. Serial Numbers 52
11.6. Sequencing Sources of Bootstrapping Data 52
12. IANA Considerations 52
12.1. The BOOTP Manufacturer Extensions and DHCP Options Registry 52
12.2. The IETF XML Registry 53
12.3. The YANG Module Names Registry 54
13. Other Considerations 54
14. Acknowledgements 54
15. References 54
15.1. Normative References 54
15.2. Informative References 56
Appendix A. Change Log 58

A.1. ID to 00 58
A.2. 00 to 01 58
A.3. 01 to 02 58
A.4. 02 to 03 59
A.5. 03 to 04 59
A.6. 04 to 05 59
A.7. 05 to 06 60
A.8. 06 to 07 60
A.9. 07 to 08 60
A.10. 08 to 09 60
A.11. 09 to 10 60
A.12. 10 to 11 61
A.13. 11 to 12 61
A.14. 12 to 13 61
Authors' Addresses 62

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 a bootstrapping strategy 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 [RFC8040] connection to the deployment specific network management system (NMS).

1.1. Use Cases

- Device connecting to a remotely administered network

  This use-case involves scenarios, such as a remote branch office or convenience store, whereby a device connects as an access gateway to an ISP's network. Assuming it is not possible to customize the ISP's network to provide any bootstrapping support, and with no other nearby device to leverage, the device has no recourse but to reach out to an Internet-based bootstrap server to bootstrap off of.

- Device connecting to a locally administered network

  This use-case covers all other scenarios and differs only in that the device may additionally leverage nearby devices, which may direct it to use a local service to bootstrap 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

This document uses the following terms:

Artifact: The term "artifact" is used throughout to represent the any of the three artifacts defined in Section 4. These artifacts collectively provide all the bootstrapping data a device needs.

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. Specifically, it refers to the artifacts defined in Section 4.

Bootstrap Information: The term "bootstrap information" is used herein to refer to one of the bootstrapping artifacts defined in Section 4. Specifically, bootstrap information is the bootstrapping data that guides a device to, for instance, install a specific boot-image and commit a specific configuration.

Bootstrap Server: The term "bootstrap server" is used within this document to mean any RESTCONF server implementing the YANG module defined in Section 10.3.

Device: The term "device" is used throughout this document to refer to the network element that needs to be bootstrapped. See Section 8 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. This identifier is used in this document to enable a Bootstrap Server to securely identify and authenticate a device.

Manufacturer: The term "manufacturer is used herein to refer to the manufacturer of a device or a delegate of the manufacturer.

Network Management System (NMS): The acronym "NMS" is used throughout this document to refer to the deployment specific management system that the bootstrapping process is responsible for introducing devices to. From a device’s perspective, when the bootstrapping process has completed, the NMS is a NETCONF or RESTCONF client.

Owner: See Rightful Owner.

Redirect Information: The term "bootstrap information" is used herein to refer to one of the bootstrapping artifacts defined in Section 4. Specifically, redirect information is the bootstrapping data that directs a device to connect to a bootstrap server.

Redirect Server: The term "redirect server" is used to refer to a subset of bootstrap servers that only returns redirect information. A redirect server is particularly useful when hosted by a manufacturer, to redirect devices to deployment-specific bootstrap servers.

Rightful Owner: The term "rightful owner" is used herein to refer to the person or organization that purchased or otherwise owns a device. Ownership is further 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.

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.

1.3. Requirements Language

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].

1.4. Tree Diagram Notation

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 (e.g., 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 ownership voucher artifact described in Section 4.3.

Using the first type of trust anchor, trust is conveyed by the device first authenticating the server (e.g., a 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 that an artifact has been signed by its rightful owner, and thereby can trust any information held within the artifact.

Notably, redirect information, as described in Section 3.1, may include more trust anchors, which illustrates another way in which trust can be conveyed.

2.3. Conveying Ownership

The ultimate goal of this document is to enable a device to establish a secure connection with its rightful owner's NMS. This entails the manufacturer being able to track who is the rightful owner of a device (not defined in this document), as well as an ability to convey that information to devices (defined in this document).

Matching the two ways to convey trust (Section 2.2), this document provides two ways to convey ownership, by using a trusted bootstrap server (Section 6.4) or by using an ownership voucher (Section 4.3).

When a device connects to a trusted bootstrap server, one that was preconfigured 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 bootstrapping data, it can authenticate that its rightful owner provided the data by verifying the signature over the data using an additional artifact defined within this document, the ownership voucher. With this approach, ownership is conveyed by the manufacturer (or delegate) taking the onus of ensuring that the ownership vouchers it issues are accurate.

3. Types of Zero Touch Information

This document defines two types of information that devices access during the bootstrapping process. These information types are described in this section.

3.1. Redirect Information

Redirect information provides information to redirect a device to a bootstrap server. Redirect information encodes a list of bootstrap servers, each defined by its hostname or IP address, an optional port, and an optional trust anchor certificate.

Redirect information is YANG modeled data formally defined by the "redirect-information" grouping in the YANG module presented in

Section 9.3. This grouping has the tree diagram shown below. Please see Section 1.4 for tree diagram notation.

```
  +--:(redirect-information)
     +--ro redirect-information
        +--ro bootstrap-server* [address]
           +--ro address         inet:host
           +--ro port?           inet:port-number
           +--ro trust-anchor?   binary
```

Redirect information MAY be trusted or untrusted. The redirect information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's rightful owner. In all other cases, the redirect information is untrusted.

Trusted redirect information is useful for enabling a device to establish a secure connection to a bootstrap server, which is possible when the redirect information includes the bootstrap server's trust anchor certificate. When a device is able to establish a secure connection to a bootstrap server, the bootstrapping data does not have to be signed in order to be trusted, as described in Section 2.2.

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 untrusted, the device MUST discard any potentially included trust anchor certificates. When the redirect information is untrusted, a device MAY establish a provisional connection to any of the specified bootstrap servers. A provisional connection is accomplished by the device blindly accepting the bootstrap server's TLS certificate. In this case, the device MUST NOT trust the bootstrap server, and data provided by the bootstrap server MUST be signed for it to be of any use to the device.

How devices process redirect information is described more formally in Section 8.5.

3.2. Bootstrap Information

Bootstrap information provides all the data necessary for a device to bootstrap itself, in order to be considered ready to be managed (e.g., by an NMS). As defined in this document, this data includes information about a boot image the device MUST be running, an initial configuration the device MUST commit, and optional scripts that, if specified, the device MUST successfully execute.

Bootstrap information is YANG modeled data formally defined by the "bootstrap-information" grouping in the YANG module presented in Section 9.3. This grouping has the tree diagram shown below. Please see Section 1.4 for tree diagram notation.

```
  +--:(bootstrap-information)
     +--ro bootstrap-information
        +--ro boot-image
        |  +--ro name       string
        |  +--ro (hash-algorithm)
        |  |  +--:(sha256)
        |  |     +--ro sha256?    string
        |  +--ro uri*       inet:uri
        +--ro configuration-handling       enumeration
        +--ro pre-configuration-script?    script
        +--ro configuration?
        +--ro post-configuration-script?   script
```

Bootstrap information MUST be trusted for it to be of any use to a device. There is no option for a device to process untrusted bootstrap information.

Bootstrap information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device’s rightful owner. In all other cases, the bootstrap information is untrusted.

How devices process bootstrap information is described more formally in Section 8.6.

4. Artifacts

This document defines three artifacts that can be made available to devices while they are bootstrapping. As will be seen in Section 6, each source of bootstrapping information specifies a means for providing each of the artifacts defined in this section.

4.1. Zero Touch Information

The information artifact encodes the essential bootstrapping data for the device. This artifact is used to encode the redirect information and bootstrap information types discussed in Section 3.

The information artifact is a PKCS#7 SignedData structure, as specified by Section 9.1 of [RFC2315], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.

Regardless how the information artifact is conveyed, the PKCS#7 structure MUST contain JSON-encoded content conforming to the YANG module specified in Section 9.3.

When the information artifact is conveyed over an untrusted transport (Section 2.2), the PKCS#7 structure structure MUST also contain a 'signerInfo' structure, as described in Section 9.1 of [RFC2315], containing a signature generated over the content using the private key associated with the owner certificate (Section 4.2).

4.2. Owner Certificate

The owner certificate artifact is a certificate that is used to identify an 'owner' (e.g., an organization), as known to a trusted certificate authority. The owner certificate is signed by a trusted certificate authority (CA), whose certificate is placed into the ownership voucher (Section 4.3).

The owner certificate is used by a device to verify the signature attached to the information artifact (Section 4.1) that the device SHOULD have also received, as described in Section 5. In particular, the device verifies signature using the public key in the owner certificate over the content contained within the information artifact.

In order to validate the owner certificate, a device MUST verify that the owner certificate's certificate chain includes the certificate specified by the ownership voucher (Section 4.3) that the device SHOULD have also received, as described in Section 5, and the device MUST verify that owner certificate contains an identifier matching the one specified in the voucher and, for devices that insist on verifying certificate revocation status, the device MUST verify that the certificate has neither expired nor been revoked.

The owner certificate artifact is formally an unsigned PKCS #7 SignedData structure as specified by Section 9.1 in [RFC2315], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.

The owner certificate artifact MUST contain the owner certificate itself and all intermediate certificates leading up to the trust anchor certificate specified in the ownership voucher. The owner certificate artifact MAY optionally include the trust anchor certificate.

Additionally, if needed by the device, the owner certificate artifact MAY also contain suitably fresh CRLs [RFC5280] and/or OCSP Responses [RFC6960].

4.3. Ownership Voucher

The ownership voucher artifact is used to securely identify a device's owner, as it is known to the manufacturer. The ownership voucher is signed by the device's manufacturer or delegate.

The ownership voucher is used by a device to verify the owner certificate (Section 4.2) that the device SHOULD have also received, as described in Section 5. In particular, the device verifies that the owner certificate's chain of trust includes the trusted certificate included in the voucher, and the device also verifies that the owner certificate contains an identifier matching the one specified in the voucher.

In order to validate the voucher, a device MUST verify that the voucher was signed by the private key associated with a trusted certificate known to the device in its factory default state, as described in Section 8.1, and the device MUST verify that the voucher includes the device's unique identifier (e.g., serial number) and, if the voucher contains an expiration date, the device MUST also verify that the voucher has not expired.

The ownership voucher artifact, including its encoding, is formally defined in [I-D.ietf-anima-voucher].

5. Artifact Groupings

Section 4 lists all the possible bootstrapping artifacts, but only certain groupings of these artifacts make sense to return in the various bootstrapping situations described in this document. The remainder of this section identifies these groupings to further clarify how the artifacts are used.

5.1. Unsigned Information

The first grouping of artifacts is for unsigned information. That is, when the information artifact (Section 4.1) has not been signed.

Unsigned information is useful for cases when transport level security can be used to convey trust (e.g., HTTPS), or when the information can be processed in a provisional manner (i.e. unsigned redirect information).

Conveying unsigned information entails communicating just one of the three artifacts listed in Section 4 as follows:

List of artifacts included in this grouping:
- zero touch information (with no embedded signature)

5.2. Signed Information (without Revocations)

The second grouping of artifacts is for when the information artifact (Section 4.1) has been signed, without any revocation information.

Signed information is needed when the information is obtained from an untrusted source of bootstrapping data (Section 6) and yet it is desired that the device be able to trust the information (i.e. no provisional processing).

Revocation information may not need to be provided because, for instance, the device only uses revocation information obtained dynamically from Internet based resources. Another possible reason may be because the device does not have a reliable clock, and therefore the manufacturer decides to never revoke information (e.g., ownership assignments are forever).

Conveying signed information without revocation information entails communicating all three of the artifacts listed in Section 4 as follows:

List of artifacts included in this grouping:
- zero touch information (with an embedded signature)
- owner certificate (with no revocation structures)
- ownership voucher

5.3. Signed Information (with Revocations)

The third grouping of artifacts is for when the information artifact (Section 4.1) has been signed and also includes revocation information.

Signed information, as described above, is needed when the information is obtained from an untrusted source of bootstrapping data (Section 6) and yet it is desired that the device be able to trust the information (i.e. no provisional processing).

Revocation information may need to be provided because, for instance, the device insists on being able to verify revocations and the device is deployed on a private network and therefore unable to obtain the revocation information from Internet based resources.

Conveying signed information with revocation information entails communicating all three of the artifacts listed in Section 4 as follows:

List of artifacts included in this grouping:
- zero touch information (with an embedded signature)
- owner certificate (with revocation structures)
- ownership voucher

6. Sources of Bootstrapping Data

This section defines some sources for zero touch bootstrapping data that a device can access. The list of sources defined here is not meant to be exhaustive. It is left to future documents to define additional sources for obtaining zero touch bootstrapping data.

For each source defined in this section, details are given for how each of the three artifacts listed in Section 4 is provided.

6.1. Removable Storage

A directly attached removable storage device (e.g., a USB flash drive) MAY be used as a source of zero touch bootstrapping data.

To use a removable storage device as a source of bootstrapping data, a device need only detect if the removable storage device is plugged in and mount its filesystem.

Use of a removable storage device is compelling, as it doesn’t require any external infrastructure to work. It is also compelling that the raw boot image file can be located on the removable storage device, enabling a removable storage device to be a fully self-standing bootstrapping solution.

A removable storage device is an untrusted source of bootstrapping data. This means that the information stored on the removable storage device either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a removable storage device presents itself as a filesystem, the bootstrapping artifacts need to be presented as files. The three artifacts defined in Section 4 are mapped to files below.

Artifact to File Mapping:

Information: Mapped to a file containing the binary artifact described in Section 4.1.

Owner Certificate: Mapped to a file containing the binary artifact described in Section 4.2.

Ownership Voucher: Mapped to a file containing the binary artifact described in Section 4.3.

The format of the removable storage device's filesystem and the naming of the files are outside the scope of this document. However, in order to facilitate interoperability, it is RECOMMENDED devices support open and/or standards based filesystems. It is also RECOMMENDED that devices assume a file naming convention that enables more than one instance of bootstrapping data to exist on a removable storage device. The file naming convention SHOULD be unique to the manufacturer, in order to enable bootstrapping data from multiple manufacturers to exist on a removable storage device.

6.2. DNS Server

A DNS server MAY be used as a source of zero touch bootstrapping data.

Using a DNS server may be a compelling option for deployments having existing DNS infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

To use a DNS server as a source of bootstrapping data, a device MAY perform a multicast DNS [RFC6762] query searching for the service "_zerotouch._tcp.local.". Alternatively the device MAY perform DNS-SD [RFC6763] via normal DNS operation, using the domain returned to it from the DHCP server; for example, searching for the service "_zerotouch._tcp.example.com".

Unsigned DNS records (not using DNSSEC as described in [RFC6698]) are an untrusted source of bootstrapping data. This means that the information stored in the DNS records either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a DNS server presents resource records (Section 3.2.1 of [RFC1035]), the bootstrapping artifacts need to be presented as resource records. The three artifacts defined in Section 4 are mapped to resource records below.

Artifact to Resource Record Mapping:

Information: Mapped to a TXT record called "zt-info" containing the base64-encoding of the binary artifact described in Section 4.1.

Owner Certificate: Mapped to a TXT record called "zt-cert" containing the base64-encoding of the binary artifact described in Section 4.2.

Ownership Voucher: Mapped to a TXT record called "zt-voucher" containing the base64-encoding of the binary artifact described in Section 4.3.

TXT records have an upper size limit of 65535 bytes (Section 3.2.1 in RFC1035), since 'RDLENGTH' is a 16-bit field. Please see Section 3.1.3 in RFC4408 for how a TXT record can achieve this size. Due to this size limitation, some information artifacts may not fit. In particular, the bootstrap information artifact could hit this upper bound, depending on the size of the included configuration and scripts.

When bootstrap information is provided, it is notable that the URL for the boot-image the device can download would have to point to another server (e.g., http://, ftp://, etc.), as DNS servers do not themselves distribute files.

6.3. DHCP Server

A DHCP server MAY be used as a source of zero touch bootstrapping data.

To use a DHCP server as a source of bootstrapping data, a device need only send a DHCP lease request to a DHCP server. However, the device SHOULD pass the Vendor Class Identifier (option 60) field in its DHCP lease request, so the DHCP server can return bootstrap information shared by devices from the same vendor. 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 the DHCP server can select the specific bootstrap information that has been staged for that one device.

Using a DHCP server may be a compelling option for deployments having existing DHCP infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

A DHCP server is an untrusted source of bootstrapping data. This means that the information returned by the DHCP server either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a DHCP server presents data as DHCP options , the bootstrapping artifacts need to be presented as

DHCP options, specifically the ones specified in Section 12.1. The three artifacts defined in Section 4 are mapped to the DHCP options specified in Section 12.1 below.

Artifact to DHCP Option Field Mapping:

Information: Mapped to the DHCP option field "zerotouch-information" containing the binary artifact described in Section 4.1.

Owner Certificate: Mapped to the DHCP option field "owner-certificate" containing the binary artifact described in Section 4.2.

Ownership Voucher: Mapped to the DHCP option field "ownership-voucher" containing the binary artifact described in Section 4.3.

When bootstrap information is provided, it is notable that the URL for the boot-image the device can download would have to point to another server (e.g., http://, ftp://, etc.), as DHCP servers do not themselves distribute files.

6.4. Bootstrap Server

A bootstrap server MAY be used as a source of zero touch bootstrapping data. A bootstrap server is defined as a RESTCONF [RFC8040] server implementing the YANG module provided in Section 10.

Unlike any other source of bootstrap data described in this document, a bootstrap server is not only a source of data, but it can also receive data from devices using the YANG-defined "notification" action statement defined in the YANG module (Section 10.3). The data sent from devices both enables visibility into the bootstrapping process (e.g., warnings and errors) as well as provides potentially useful completion status information (e.g., the device’s SSH host-keys).

To use a bootstrap server as a source of bootstrapping data, a device MUST use the RESTCONF protocol to access the YANG container node /device/, passing its own serial number in the URL as the key to the ’device’ list.

Using a bootstrap server as a source of bootstrapping data is a compelling option as it uses transport-level security in lieu of signed data, which may be easier to deploy in some situations. Additionally, the bootstrap server is able to receive notifications

from devices, which may be critical to some deployments (e.g., the passing of the device's SSH host keys).

A bootstrap server may be trusted or an untrusted source of bootstrapping data, depending on how the device learned about the bootstrap server's trust anchor from a trusted source. When a bootstrap server is trusted, the information returned from it MAY be signed. However, when the server is untrusted, in order for its information to be of any use to the device, the information MUST either be signed or be information that can be processed provisionally (e.g., unsigned redirect information).

When a device is able to trust a bootstrap server, it MUST send its IDevID certificate in the form of a TLS client certificate, and it MUST send notifications to the bootstrap server. When a device is not able to trust a bootstrap server, it MUST NOT send its IDevID certificate in the form of a TLS client certificate, and it MUST NOT send any notifications to the bootstrap server.

From an artifact perspective, since a bootstrap server presents data as a YANG-modeled data, the bootstrapping artifacts need to be mapped to nodes in the YANG module. The three artifacts defined in Section 4 are mapped to bootstrap server nodes defined in Section 10.3 below.

Artifact to Bootstrap Server Node Mapping:

Information: Mapped to the node /device/zerotouch-information.

Owner Certificate: Mapped to the leaf node /device/owner-certificate.

Ownership Voucher: Mapped to the leaf node /device/ownership-voucher.

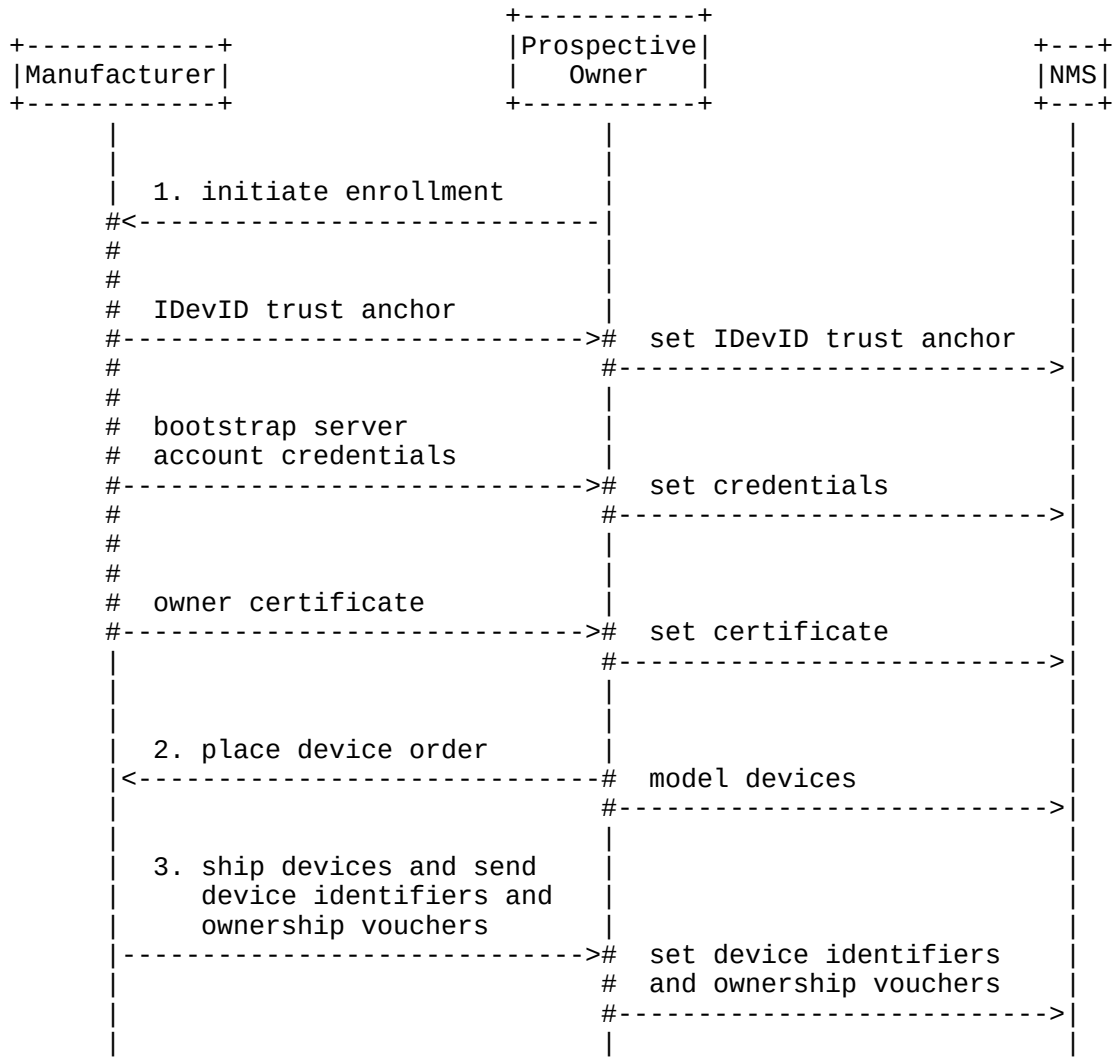
While RESTCONF servers typically support a nested hierarchy of resources, zero touch bootstrap servers only need to support the paths /device and /device/notification. The device processing instructions provided in Section 8.3 only uses these two URLs.

7. 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.

7.1. Onboarding and Ordering Devices

The following diagram illustrates key interactions that may 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.



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

1. A prospective owner of a manufacturer's devices, or an existing owner that wishes to start using zero touch for future device orders, initiates an enrollment process with the manufacturer or delegate. This process includes the following:

   * Regardless how the prospective owner intends to bootstrap their devices, they will always obtain from the manufacturer or delegate the trust anchor certificate for its device's 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 (e.g., a redirect server) such as described in Section 6.4, 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 validate signed data (Section 8.4), then the manufacturer, acting as a certificate authority, may additionally sign an owner certificate for the prospective owner. Alternatively, and not depicted, the owner may obtain an owner certificate from a manufacturer-trusted 3rd-party certificate authority, and report that certificate to the manufacturer. How the owner certificate is used to enable devices to validate signed bootstrapping data is described in Section 8.4. 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.

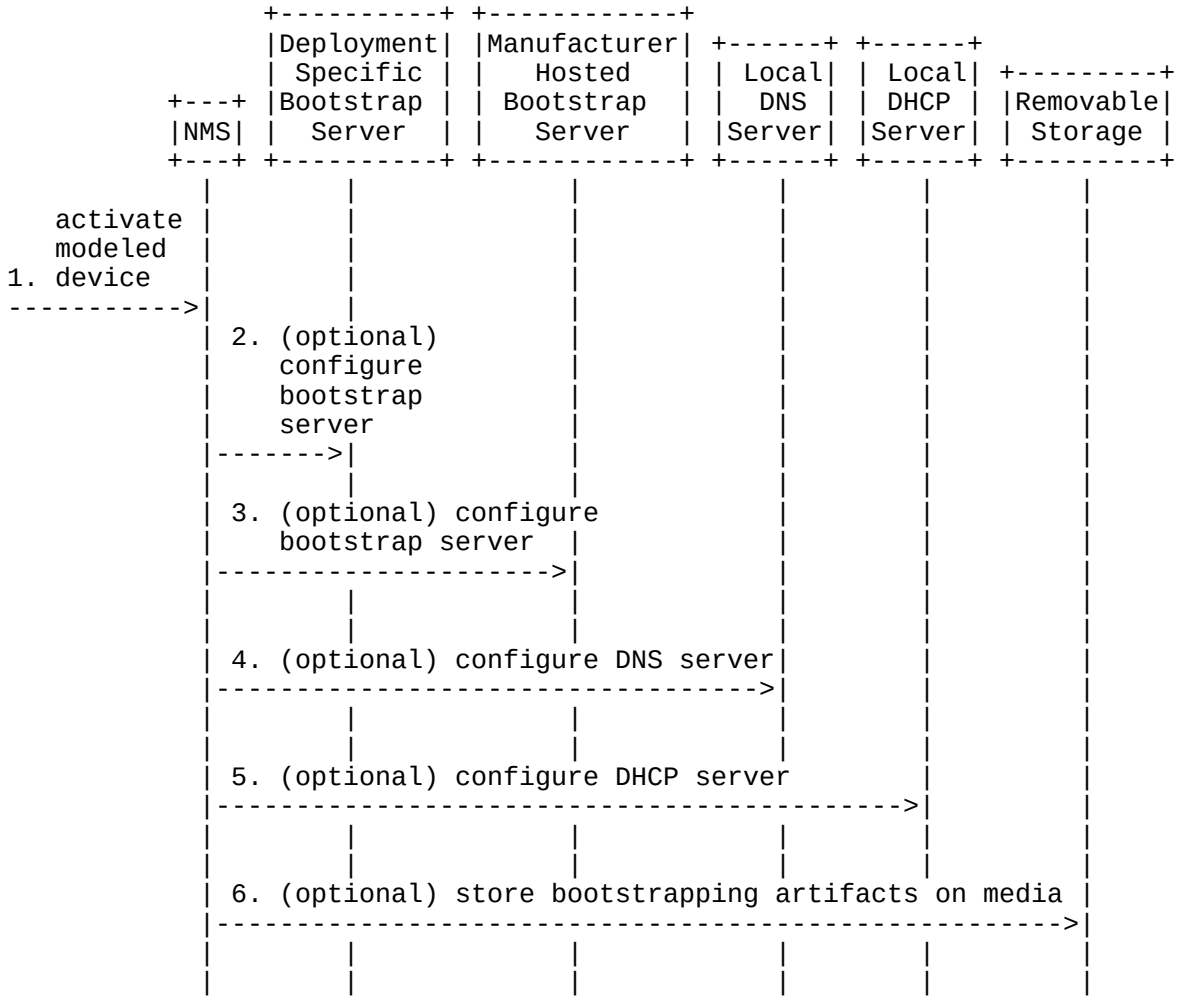
2. Some time later, the prospective owner places an order with the manufacturer (or delegate), perhaps with a special flag checked for zero touch handling. At this time, or perhaps before placing the order, the owner may model the devices in their NMS, creating virtual objects for the devices with no real-world device associations. For instance the model can be used to simulate the device's location in the network and the configuration it should have when fully operational.

3. When the manufacturer or delegate fulfills the order, shipping the devices to their intended locations, they may notify the owner of the devices's unique identifiers (e.g., serial numbers) and shipping destinations, which the owner may use to stage the network for when the devices power on. Additionally, the manufacturer may send one or more ownership vouchers,

cryptographically assigning ownership of those devices to the rightful owner. The owner may set this information on their NMS, perhaps binding specific modeled devices to the unique identifiers and ownership vouchers.

7.2. Owner Stages the Network for Bootstrap

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



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

1. Having previously modeled the devices, including setting their fully operational configurations and associating both device identifiers (e.g., serial numbers) and ownership vouchers, the owner "activates" one or more modeled devices. That is, the owner tells the NMS to perform the steps necessary to prepare for when the real-world devices power up and initiate the bootstrapping process. Note that, in some deployments, this step might be combined with the last step from the previous workflow. Here it is depicted that an NMS performs the steps, but they may be performed manually or through some other mechanism.

2. If it is desired to use a deployment specific bootstrap server, it MUST be configured to provide the bootstrapping information for the specific devices. Configuring the bootstrap server MAY occur via a programmatic API not defined by this document. Illustrated here as an external component, the bootstrap server MAY be implemented as an internal component of the NMS itself.

3. If it is desired to use a manufacturer (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.

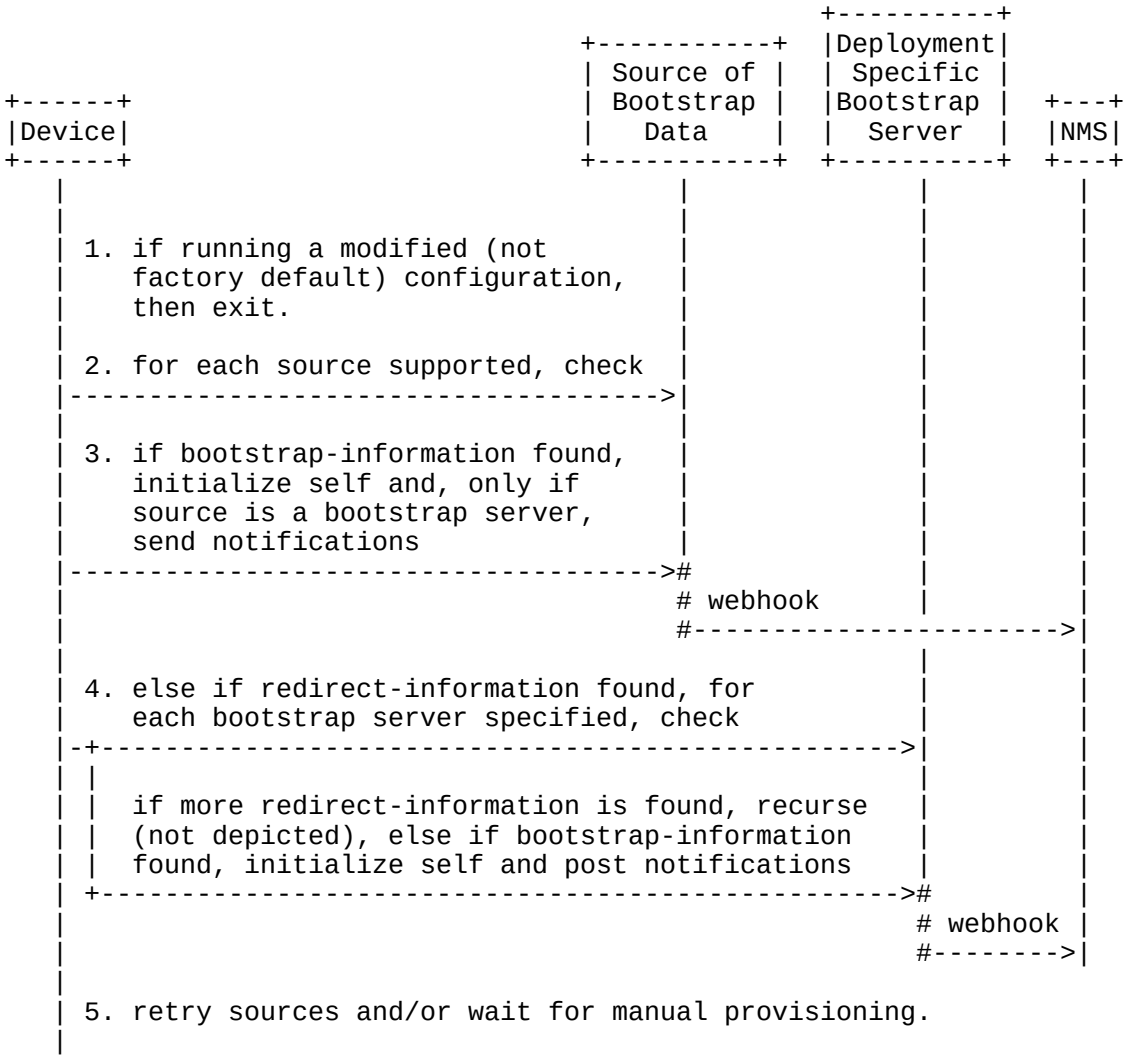
4. If it is desired to use a DNS server to supply bootstrapping information, a DNS server needs to be configured. If multicast DNS-SD is desired, then the server MUST reside on the local network, otherwise the DNS server MAY reside on a remote network. Please see Section 6.2 for more information about how to configure DNS servers. Configuring the DNS server MAY occur via a programmatic API not defined by this document.

5. If it is desired to use a DHCP server to supply bootstrapping data, a DHCP server needs to be configured. The DHCP server may be accessed directly or via a DHCP relay. Please see Section 6.3 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 6.1 for more information about how to configure a removable storage device.

7.3. Device Powers On

The following diagram illustrates the sequence of activities that occur when a device powers on.



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 is in a modified state, then the bootstrapping logic exits and none of the following interactions occur.

2. For each source of bootstrapping data the device supports, preferably in order of closeness to the device (e.g., removable storage before Internet based servers), the device checks to see if there is any bootstrapping data for it there.

3. If bootstrap-information is found, the device initializes itself accordingly (e.g., installing a boot-image and committing an initial configuration). If the source is a bootstrap server, and the bootstrap server can be trusted (i.e., TLS-level authentication), the device also sends progress notifications to the bootstrap server.

   * The contents of the initial configuration SHOULD configure an administrator account on the device (e.g., username, ssh-rsa key, etc.) and SHOULD configure the device either to listen for NETCONF or RESTCONF connections or to initiate call home connections [RFC8071].

   * If the bootstrap server supports forwarding device notifications to external systems (e.g., via a webhook), the "bootstrap-complete" notification (Section 10.3) informs the external system to know when it can, for instance, initiate a connection to the device (assuming it knows the device's address and the device was configured to listen for connections). To support this further, the bootstrap-complete notification also relays the device's SSH host keys and/or TLS certificates, with which the external system can use to authenticate subsequent connections to the device.

   If the device is ever able to complete the bootstrapping process successfully (i.e., no longer running its factory default configuration), it exits the bootstrapping logic without considering any additional sources of bootstrapping data.

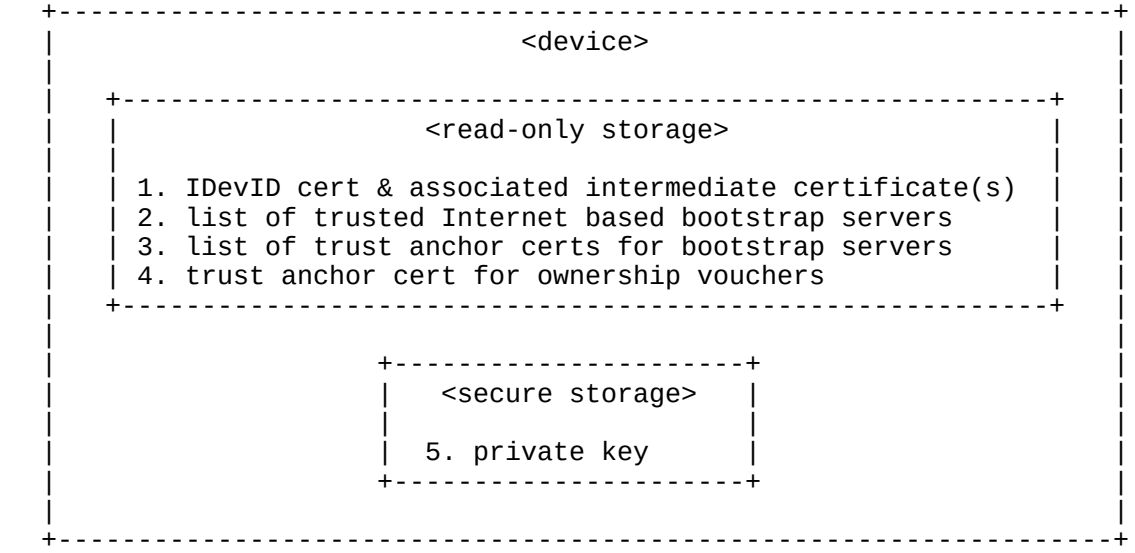
4. Otherwise, if redirect-information is found, the device iterates through the list of specified bootstrap servers, checking to see if there is any bootstrapping data for it on them. If the bootstrap server returns more redirect-information, then the device processes it recursively. Otherwise, if the bootstrap server returns bootstrap-information, the device processes it following the description provided in (3) above.

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

8. Device Details

Devices supporting the bootstrapping strategy described in this document MUST have the preconfigured factory default state and bootstrapping logic described in the following sections.

8.1. Factory Default State



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

1. 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 the device's 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, which is provided to prospective owners separately (e.g., Section 7.1).

2. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 6.4) MUST be manufactured with a configured list of trusted bootstrap servers. Consistent with redirect information (Section 3.1, each bootstrap server MAY be identified by its hostname or IP address, and an optional port.

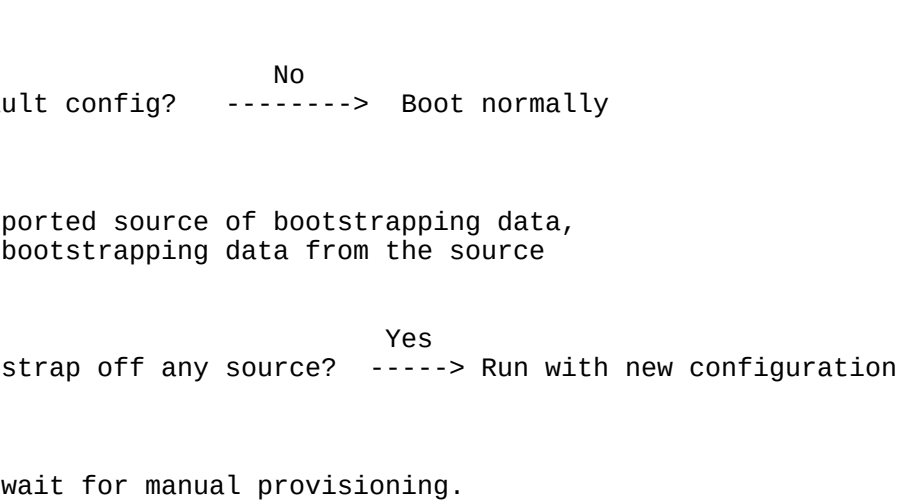
3. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 6.4) MUST also be manufactured with a list of trust anchor certificates that can be used for X.509 certificate path validation ([RFC6125], Section 6) on the bootstrap server's TLS server certificate.

4. Devices that support loading owner signed data (see Section 1.2) MUST also be manufactured with the trust anchor certificate for the ownership vouchers.

5. 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).

8.2. Boot Sequence

A device claiming to support the bootstrapping strategy defined in this document MUST support the boot sequence described in this section.



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

1. When the device powers on, it first checks to see if 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 6). Details for how to processes a source of bootstrapping data are provided in Section 8.3.

3. If the device is able to bootstrap itself off any of the sources of 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.

8.3. Processing a Source of Bootstrapping Data

This section describes a recursive algorithm that a device claiming to support the bootstrapping strategy defined in this document 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 (Section 3.2) 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 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 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. If trust-state is TRUE, then the device MAY also accesses the bootstrap servers 'notification' resource for the device.

For any source of bootstrapping data (e.g., Section 6), if the data is signed and the device is able to validate the signed data using the algorithm described in Section 8.4, 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 8.2. Otherwise, the device MUST attempt to process the bootstrap information as described in Section 8.6. In either case, success or failure, the device MUST exit the recursive algorithm, returning to the state machine described in Section 8.2, 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 8.5. 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.

8.4. Validating Signed Data

Whenever a device is presented signed data from an untrusted source, it MUST validate the signed data as described in this section. If the signed data is provided by a trusted source, a redundant trust case, the device MAY skip verifying the signature.

Whenever there is signed data, the device MUST also be provided an ownership voucher and an owner certificate. Depending on circumstances, the device MAY also be provided certificate revocations. How all the needed artifacts are provided for each source of bootstrapping data is defined in Section 6.

The device MUST first authenticate the ownership voucher by validating the signature on it to one of its preconfigured trust anchors (see Section 8.1) and verify that the voucher contains the device's unique identifier (e.g., serial number). If the voucher contains an expiration timestamp, the device MUST also verify that the voucher has not expired. If the authentication of the voucher is successful, the device extracts from it information that can be used to verify the owner certificate in the next step.

Next the device MUST authenticate the owner certificate by performing X.509 certificate path verification to the trusted certificate provided in the voucher. If the device insists on verifying revocation status, it MUST also verify that none of the certificates in the chain of certificates have been revoked or expired. 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 verify the signature over information artifact 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.

8.5. 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. The device sequentially steps through the list of provided bootstrap servers until it can find one it can bootstrap off of.

If a hostname is provided, and the hostname's DNS resolution is to more than one IP address, the device MUST attempt to connect to all of the DNS resolved addresses at least once, before moving on to the next bootstrap server. If the device is able to obtain bootstrapping data from any of the DNS resolved addresses, it MUST immediately process that data, without attempting to connect to any of the other DNS resolved addresses.

If the redirect information is trusted (e.g., trust-state is TRUE), and the bootstrap server entry contains a trust anchor certificate, then the device MUST authenticate the bootstrap server using X.509 certificate path validation ([RFC6125], Section 6) to the specified trust anchor. If the device is unable to authenticate the bootstrap server to the specified trust anchor, the device MUST NOT attempt a provisional connection to the bootstrap server (i.e., by blindly accepting its server certificate).

If the redirect information is untrusted (e.g., trust-state is FALSE), the device MUST discard any trust anchors provided by the redirect information and establish a provisional connection to the bootstrap server (i.e., by blindly accepting its TLS server certificate).

8.6. 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 execute the pre-configuration script (if any), 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 (e.g., name or fingerprint match). If it does not, the device MUST download, verify, and install the specified boot image, and then reboot. To verify the boot image, the device MUST check that the boot image file matches the fingerprint (e.g., sha256) 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 will move to processing the next step.

Next, for devices that support executing scripts, if a pre-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if had any warnings or 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.

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

Again, for devices that support executing scripts, if a post-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if it had any warnings or 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 device obtained the bootstrap information from a trusted bootstrap server, the device MUST send the 'bootstrap-complete' notification now.

At this point the device is configured and no longer running its factory default configuration. Notably, if the bootstrap information

configured the device it initiate a call home connection, the device would proceed to do so now.

9. The Zero Touch Information Artifact

This section defines a YANG [RFC6020] module that is used to define the data model for the Zero Touch Information artifact described in Section 4.1. Examples illustrating this artifact in use are provided in Section 9.2.

9.1. Tree Diagram

The following tree diagram provides an overview of the data model for the Zero Touch Information artifact. The syntax used for this tree diagram is described in Section 1.4.

```
module: ietf-zerotouch-information
  +---- (information-type)
     +--:(redirect-information)
     |  +---- redirect-information
     |     +---- bootstrap-server* [address]
     |        +---- address         inet:host
     |        +---- port?           inet:port-number
     |        +---- trust-anchor?   binary
     +--:(bootstrap-information)
        +---- bootstrap-information
           +---- boot-image
           |  +---- name      string
           |  +---- (hash-algorithm)
           |  |  +--:(sha256)
           |  |     +---- sha256?   string
           |  +---- uri*      inet:uri
           +---- configuration-handling?      enumeration
           +---- pre-configuration-script?    script
           +---- configuration?
           +---- post-configuration-script?   script
```

9.2. Example Usage

This section presents examples for how the information artifact (Section 4.1) can be encoded into a document that can be distributed outside the bootstrap server’s RESTCONF API.

The following example illustrates how redirect information can be encoded into an artifact.

```
   <redirect-information
     xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-information">
     <bootstrap-server>
       <address>phs1.example.com</address>
       <port>8443</port>
       <trust-anchor>Base64-encoded X.50 </trust-anchor>
     </bootstrap-server>
     <bootstrap-server>
       <address>phs2.example.com</address>
       <port>8443</port>
       <trust-anchor>Base64-encoded X.50 </trust-anchor>
     </bootstrap-server>
     <bootstrap-server>
       <address>phs3.example.com</address>
       <port>8443</port>
       <trust-anchor>Base64-encoded X.50 </trust-anchor>
     </bootstrap-server>
   </redirect-information>
```

The following example illustrates how bootstrap information can be encoded into an artifact. This example uses datamodels from [RFC7317] and [I-D.ietf-netconf-netconf-client-server].

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

<bootstrap-information
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-information">
  <boot-image>
    <name>boot-image-v3.2R1.6.img</name>
    <sha256>Hex-encoded SHA256 hash</sha256>
    <uri>file:///some/path/to/raw/file </uri>
  </boot-image>
  <configuration-handling>merge</configuration-handling>
  <configuration>
    <!-- from ietf-system.yang -->
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      <authentication>
        <user>
          <name>admin</name>
          <authorized-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>
          </authorized-key>
        </user>
      </authentication>
    </system>
    <!-- from ietf-netconf-server.yang -->
    <netconf-server
      xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
      <call-home>
        <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>
                <name>certificate</name>
                <certificate>builtin-idevid-cert</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-cli
ent-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>
      </call-home>
    </netconf-server>
  </configuration>
```

```
</bootstrap-information>
```

9.3. YANG Module

The Zero Touch Information artifact is normatively defined by the YANG module defined in this section.

Note: the module defined herein uses data types defined in [RFC5280], [RFC6234], and [RFC6991].

```
<CODE BEGINS> file "ietf-zerotouch-information@2017-03-13.yang"

module ietf-zerotouch-information {
  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-zerotouch-information";
  prefix    "zti";

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

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

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

  contact
   "WG Web:   http://tools.ietf.org/wg/netconf
    WG List:  <mailto:netconf@ietf.org>
    Author:   Kent Watsen <mailto:kwatsen@juniper.net>";

  description
   "This module defines the data model for the Zero Touch Information
    artifact defined by RFC XXXX: Zero Touch Provisioning for NETCONF
    or RESTCONF based Management.

    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 "2017-03-13" {
    description
     "Initial version";
    reference
     "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
      Management";
  }

  rc:yang-data zerotouch-information {
    uses zerotouch-information-grouping;
  }

  grouping zerotouch-information-grouping {
    description
      "Defines the zerotouch information data model.  Grouping
       exists only to enable pyang tree output.";

    choice information-type {
      mandatory true;
      description
        "This choice statement ensures the response only contains
         redirect-information or bootstrap-information.  Note that
         this is the only mandatory true node, as the other nodes
         are not needed when the device trusts the bootstrap server,
         in which case the data does not need to be signed.";

      container redirect-information {
        description
          "This is redirect information, as described in Section 3.1
           in RFC XXXX.  Its purpose is to redirect a device to another
           bootstrap server.";
        reference
          "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF
           based Management";

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

```
          leaf address {
            type inet:host;
            mandatory true;
            description
             "The IP address or hostname of the bootstrap server the
              device should redirect to.";
          }
          leaf port {
            type inet:port-number;
            default 443;
            description
             "The port number the bootstrap server listens on.";
          }
          leaf trust-anchor { //should there be two fields like voucher?
            type binary;
            description
              "An X.509 v3 certificate structure as specified by RFC
               5280, Section 4, encoded using 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, as described in Section 3.2 in
           RFC XXXX.  Its purpose is to provide the device everything it
           needs to bootstrap itself.";
        reference
          "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF
           based Management";

        container boot-image {
          description
            "Specifies criteria for the boot image the device MUST
```

```
             be running.";

          leaf name {  // maybe this should be a regex?
            type string;
            mandatory true;
            description
              "The name of a software image that the device MUST
               be running in order to process the remaining nodes.";
          }
          choice hash-algorithm {
             mandatory true;
             description
               "Identifies the hash algorithm used.";
             leaf sha256 {
                type string;
                description
                  "The hex-encoded SHA-256 hash over the boot
                   image file.  This is used by the device to
                   verify a downloaded boot image file.";
                reference
                  "RFC 6234: US Secure Hash Algorithms.";
             }
          }
          leaf-list uri {
            type inet:uri;
            min-elements 1;
            description
              "An ordered list of URIs to where the boot-image file MAY
               be obtained.  Deployments MUST know in which URI schemes
               (http, ftp, etc.) a device supports.  If a secure scheme
               (e.g., https) is provided, a device MAY establish a
               provisional connection to the server, by blindly
               accepting the server's credentials (e.g., its TLS
               certificate)";
          }
        }

        leaf configuration-handling {
          type enumeration {
            enum merge {
              description
               "Merge configuration into existing running configuration.";
            }
            enum replace {
              description
                "Replace existing running configuration with the passed
                 configuration.";
            }
```

```
          }
          description
            "This enumeration indicates how the server should process
             the provided configuration.  When not specified, the device
             MAY determine how to process the configuration using other
             means (e.g., vendor-specific metadata).";
        }

        leaf pre-configuration-script {
          type script;
          description
            "A script that, when present, is executed before the
             configuration has been processed.";
        }

        anydata configuration {
          must "../configuration-handling";
          description
            "Any configuration data model known to the device.  It may
             contain manufacturer-specific and/or standards-based data
             models.";
        }

        leaf post-configuration-script {
          type script;
          description
            "A script that, when present, is executed after the
             configuration has been processed.";
        }
      }
    }
  }

  typedef script {
    type binary;
    description
      "A device specific script that enables the execution of commands
       to perform actions not possible thru configuration alone.

       No attempt is made to standardize the contents, running context,
       or programming language of the script.  The contents of the
       script are considered specific to the vendor, product line,
       and/or model of the device.

       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 returns 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 a soft error, which 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, which the script believes
        will affect manageability.  In this case, the device SHOULD
        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.";
    }

}

<CODE ENDS>
```

10. The Zero Touch Bootstrap Server API

This section defines a YANG [RFC6020] module that is used to define the RESTCONF [RFC8040] API used by the bootstrap server defined in Section 6.4. Examples illustrating this API in use are provided in Section 10.2.

10.1. Tree Diagram

The following tree diagram provides an overview for the bootstrap server RESTCONF API. The syntax used for this tree diagram is described in Section 1.4.

```
module: ietf-zerotouch-bootstrap-server
    +--ro device* [unique-id]
       +--ro unique-id                string
       +--ro zerotouch-information    pkcs7
       +--ro owner-certificate?       pkcs7
       +--ro ownership-voucher?       pkcs7
       +---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    pkcs7
```

In the above diagram, notice that all of the protocol accessible nodes 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 use to provide progress notifications to the bootstrap server.

10.2. Example Usage

This section presents some examples illustrating the bootstrap server's API. Two examples are provided, one illustrating a device fetching bootstrapping data from the server, and the other illustrating a data posting a progress notification to the server.

The following example illustrates a device using the API to fetch its bootstrapping data from the bootstrap server. In this example, the device receives a signed response; an unsigned response would look similar except the last two fields (owner-certificate and ownership-voucher) would be absent in the response.

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

GET https://example.com/restconf/data/ietf-zerotouch-bootstrap-server:\
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>
  <zerotouch-information>\
    Base64-encoded Zero Touch Information artifact\
  <zerotouch-information>
  <owner-certificate>Base64-encoded PKCS#7</owner-certificate>
  <ownership-voucher>Base64-encoded Voucher</ownership-voucher>
</device>
```

The following example illustrates a device using the API to post a notification to a bootstrap server. Illustrated below is the 'bootstrap-complete' message, but the device may send other notifications to the server while bootstrapping (e.g., to provide status updates). In this message, the device is sending both its SSH host keys and TLS server certificate, which the bootstrap server may, for example, pass to an NMS, as discussed in Section 7.3.

Note that devices that are able to present an IDevID certificate [Std-802.1AR-2009] when establishing SSH or TLS connections do not need to include its DevID certificate in the bootstrap-complete message. It is unnecessary to send the DevID certificate in this case because the IDevID certificate does not need to be pinned by an NMS in order to be trusted.

Note that 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 (e.g., when sending signed information).

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

POST https://example.com/restconf/data/ietf-zerotouch:\
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>Base64-encoded SSH RSA Public Key</key-data>
    </ssh-host-key>
    <ssh-host-key>
      <format>ssh-dsa</format>
      <key-data>Base64-encoded SSH DSA Public Key</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>Base64-encoded X.509</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
```

10.3. YANG Module

The bootstrap server's device-facing API is normatively defined by the YANG module defined in this section.

Note: the module defined herein uses data types defined in [RFC2315], [RFC5280], and [I-D.ietf-anima-voucher].

<CODE BEGINS> file "ietf-zerotouch-bootstrap-server@2017-03-13.yang"

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

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

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

  contact
   "WG Web:   <http://tools.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>
    Author:   Kent Watsen
              <mailto:kwatsen@juniper.net>";

  description
   "This module defines an interface for bootstrap servers, as defined
    by RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
    Management.

    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 "2017-03-13" {
    description
     "Initial version";
    reference
     "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
```

```
      Management";
  }

  // typedefs

  typedef pkcs7 {
    type binary;
    description
      "A PKCS #7 SignedData structure, as specified by Section 9.1
       in RFC 2315, encoded using ASN.1 distinguished encoding rules
       (DER), as specified in ITU-T X.690.";
     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).";
  }

  // protocol accessible node

  list device {
    key unique-id;
    config false;

    description
      "A device's record entry.  This is the only RESTCONF resource
       that a device will GET, as described in Section 8.2 in RFC XXXX.
       Getting just this top-level node provides a device with all the
       data it needs in a single request.";
    reference
      "RFC XXXX: Zero Touch Provisioning for NETCONF or
       RESTCONF based Management";

    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.";
    }

    leaf zerotouch-information {
      type pkcs7;
      mandatory true;
      description
```

```
        "A 'zerotouch-information' artifact, as described in Section
         4.1 of RFC XXXX.  When conveyed over an untrusted transport, in
         order to be processed by a device, this PKCS#7 SignedData
         structure  MUST contain a 'signerInfo' structure, described
         in Section 9.1 of RFC 2315, containing a signature generated
         using the owner's private key.";
      reference
        "RFC XXXX: Zero Touch Provisioning for NETCONF or
           RESTCONF based Management.
         RFC 2315:
           PKCS #7: Cryptographic Message Syntax Version 1.5";
    }

    leaf owner-certificate {
      type pkcs7;
      must "../zerotouch-information" {
        description
          "An 'zerotouch-information' artifact must be present
           whenever an ownership certificate is presented.";
      }
      description
        "An unsigned PKCS #7 SignedData structure, as specified by
         Section 9.1 in RFC 2315, encoded using ASN.1 distinguished
         encoding rules (DER), as specified in ITU-T X.690.

         This structure MUST contain the owner certificate and all
         intermediate certificates leading up to at least the trust
         anchor certificate specified in the ownership voucher.
         Additionally, if needed by the device, this structure MAY
         also contain suitably fresh CRL and or OCSP Responses.

         X.509 certificates and CRLs are described in RFC 5280.
         OCSP Responses are described in RFC 6960.";
      reference
       "RFC 2315:
          PKCS #7: Cryptographic Message Syntax Version 1.5.
        RFC 5280:
          Internet X.509 Public Key Infrastructure Certificate
          and Certificate Revocation List (CRL) Profile.
        RFC 6960:
          X.509 Internet Public Key Infrastructure Online
          Certificate Status Protocol - OCSP.
        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 ownership-voucher {
      type pkcs7;
      must "../owner-certificate" {
        description
          "An owner certificate must be present whenever an ownership
           voucher is presented.";
      }
      description
        "A 'voucher' artifact, as described in Section 5 of
         I-D.ietf-anima-voucher.  The voucher's 'device-identifier'
         MUST reference both the device's unique identifier and
         the owner's owner certificate.";
      reference
        "I-D.etf-anima-voucher:
           Voucher and Voucher Revocation Profiles for Bootstrapping
           Protocols";
    }

    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 MAY
                 contain any additional information that the
                 manufacturer thinks might be useful.";
            }
            enum parsing-warning {
              description
                "Indicates that the device had a non-fatal error when
                 parsing the response from the bootstrap server.  The
                 'message' field below SHOULD indicate the specific
                 warning that occurred.";
            }
            enum parsing-error {
              description
                "Indicates that the device encountered a fatal error
                 when parsing the response from the bootstrap server.
                 For instance, this could be due to malformed encoding,
                 the device expecting signed data when only unsigned
                 data is provided, because the ownership voucher didn't
                 include the device's unique identifier, or because the
                 signature didn't match.  The 'message' field below
                 SHOULD indicate the specific error.  This notification
                 also indicates that the device has abandoned trying to
                 bootstrap off this bootstrap server.";
```

```
        }
        enum boot-image-warning {
          description
            "Indicates that the device encountered a non-fatal
             error condition when trying to install a boot-image.
             A possible reason might include a need to reformat a
             partition causing loss of data.  The 'message' field
             below SHOULD indicate any warning messages that were
             generated.";
        }
        enum boot-image-error {
          description
            "Indicates that the device encountered an error when
             trying to install a boot-image, which could be for
             reasons such as a file server being unreachable,
             file not found, signature mismatch, etc.  The
             'message' field SHOULD indicate the specific error
             that occurred.  This notification also indicates
             that the device has abandoned trying to bootstrap
             off this bootstrap server.";
        }
        enum pre-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, as well as
             capture any stdout/stderr messages the script may
             have produced.";
        }
        enum pre-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, as well as capture any
             stdout/stderr messages the script may have produced.
             This notification 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 any warning
             messages that were generated.";
        }
```

```
            enum config-error {
              description
                "Indicates that the device obtained error messages
                 when it committed the initial configuration.  The
                 'message' field below SHOULD indicate the error
                 messages that were generated.  This notification
                 also indicates that the device has abandoned trying
                 to bootstrap off this bootstrap server.";
            }
            enum post-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, as well as
                 capture any stdout/stderr messages the script may
                 have produced.";
            }
            enum post-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, as well as capture any
                 stdout/stderr messages the script may have produced.
                 This notification 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 MAY contain any
                 additional information that the manufacturer thinks
                 might be useful.  After sending this notification,
                 the device is not expected to access the bootstrap
                 server again.";
            }
            enum informational {
              description
                "Indicates any additional information not captured by
                 any of the other notification-type. For instance, a
                 message indicating that the device is about to reboot
                 after having installed a boot-image could be provided.
                 The 'message' field below SHOULD contain information
                 that the manufacturer thinks might be useful.";
            }
```

```
          }
          mandatory true;
          description
            "The type of notification provided.";
        }
        leaf message {
          type string;
          description
            "An optional human-readable value.";
        }
        container ssh-host-keys {
          when "../notification-type = 'bootstrap-complete'" {
            description
              "SSH host keys are only sent when the notification
               type is 'bootstrap-complete'.";
          }
          description
            "A list of SSH host keys an NMS may use to authenticate
             a NETCONF connection to the device with.";
          list ssh-host-key {
            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 {
          when "../notification-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.";
          list trust-anchor {
            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 pkcs7;
              mandatory true;
              description
                "An X.509 v3 certificate structure, as specified by
                 Section 4 in RFC5280, encoded using 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

  } // end device

}

<CODE ENDS>
```

11. Security Considerations

11.1. Immutable storage for trust anchors

Devices MUST ensure that all their trust anchor certificates, including those for connecting to bootstrap servers and verifying ownership vouchers, are protected from external modification.

It may be necessary to update these certificates over time (e.g., the manufacturer wants to delegate trust to a new CA). It is therefore expected that devices MAY update these trust anchors when needed through a verifiable process, such as a software upgrade using signed software images.

11.2. Clock Sensitivity

The solution in this document relies on TLS certificates, owner certificates, and ownership vouchers, all of which require an accurate clock in order to be processed correctly (e.g., to test validity dates and revocation status). Implementations MUST ensure devices have an accurate clock when shipped from manufacturing facilities, and take steps to prevent clock tampering.

If it is not possible to ensure clock accuracy, it is RECOMMENDED that implementations disable the aspects of the solution having clock sensitivity. In particular, such implementations should assume that TLS certificates and owner certificates are not revokable. In real-world terms, this means that manufacturers SHOULD only issue a single ownership voucher for the lifetime of some devices.

It is important to note that implementations SHOULD NOT rely on NTP for time, as it is not a secure protocol.

11.3. 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, which is desirable to support in some cases.

To compensate for this, this document requires that devices, when connected to an untrusted bootstrap server, do not send their IDevID certificate for client authentication, and they do not POST any progress notifications, and they assert that data downloaded from the server is signed.

11.4. 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 the notAfter value 99991231235959Z). Given the long-lived nature of these certificates, it is paramount to use a strong key length (e.g., 512-bit ECC).

11.5. 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.

11.6. Sequencing Sources of Bootstrapping Data

For devices supporting more than one source for bootstrapping data, no particular sequencing order has to be observed for security reasons, as the solution for each source is considered equally secure. However, from a privacy perspective, it is RECOMMENDED that devices access local sources before accessing remote sources.

12. IANA Considerations

12.1. 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.

Tag: XXX

Name: Zero Touch Information

Returns up to three zero touch bootstrapping artifacts.

```
 Code   Len
+-----+-----+-----------------------+
| XXX |  n  | zerotouch-information |
+-----+-----+-----------------------+

+-------------------+-------------------+
| owner-certificate | ownership-voucher |
+-------------------+-------------------+
```

Reference: RFC XXXX

Tag: YYY

Name: Zero Touch Information

Returns up to three zero touch bootstrapping artifacts.

```
 Code   Len
+-----+-----+-----------------------+
| XXX |  n  | zerotouch-information |
+-----+-----+-----------------------+

+-------------------+-------------------+
| owner-certificate | ownership-voucher |
+-------------------+-------------------+
```

Reference: RFC XXXX

12.2. 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
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

12.3. The YANG Module Names Registry

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

```
name:         ietf-zerotouch-information
namespace:    urn:ietf:params:xml:ns:yang:ietf-zerotouch
prefix:       zt
reference:    RFC XXXX

name:         ietf-zerotouch-bootstrap-server
namespace:    urn:ietf:params:xml:ns:yang:ietf-zerotouch
prefix:       zt
reference:    RFC XXXX
```

13. Other Considerations

Both this document and [I-D.ietf-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.

14. 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, Ian Farrer, Stephen Hanna, Wes Hardaker, Russ Mundy, Reinaldo Penno, Randy Presuhn, Max Pritikin, Michael Richardson, Phil Shafer, 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.

15. References

15.1. Normative References

[I-D.ietf-anima-voucher]
Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "Voucher and Voucher Revocation Profiles for Bootstrapping Protocols", draft-ietf-anima-voucher-00 (work in progress), January 2017.

[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>.

[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>.

[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>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

[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>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <http://www.rfc-editor.org/info/rfc7468>.

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

[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>.

15.2. Informative References

[I-D.ietf-anima-bootstrapping-keyinfra] Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-04 (work in progress), October 2016.

[I-D.ietf-netconf-netconf-client-server] Watsen, K., Wu, G., and J. Schoenwaelder, "NETCONF Client and Server Models", draft-ietf-netconf-netconf-client-server-01 (work in progress), November 2016.

[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>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <http://www.rfc-editor.org/info/rfc6960>.

[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>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <http://www.rfc-editor.org/info/rfc8071>.

Appendix A. Change Log

A.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

A.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

A.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

A.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.

A.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".

A.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

A.7. 05 to 06

- Minor update
- Added many Normative and Informative references.
- Added new section Other Considerations.

A.8. 06 to 07

- Minor update
- Added an Editorial Note section for RFC Editor.
- Updated the IANA Considerations section.

A.9. 07 to 08

- Minor update
- Updated to reflect review from Michael Richardson.

A.10. 08 to 09

- Added in missing "Signature" artifact example.
- Added recommendation for manufacturers to use interoperable formats and file naming conventions for removable storage devices.
- Added configuration-handling leaf to guide if config should be merged, replaced, or processed like an edit-config/yang-patch document.
- Added a pre-configuration script, in addition to the post-configuration script from -05 (issue #15).

A.11. 09 to 10

- Factored ownership vocher and voucher revocation to a separate document: draft-kwatsen-netconf-voucher. (issue #11)
- Removed <configuration-handling> options ’edit-config’ and yang-patch’. (issue #12)
- Defined how a signature over signed-data returned from a bootstrap server is processed. (issue #13)

- Added recommendation for removable storage devices to use open/ standard file systems when possible. (issue #14)
- Replaced notifications "script-[warning/error]" with "[pre/post]-script-[warning/error]". (goes with issue #15)
- switched owner-certificate to be encoded using the pkcs#7 format. (issue #16)
- Replaced md5/sha1 with sha256 inside a choice statement, for future extensibility. (issue #17)
- A ton of editorial changes, as I went thru the entire draft with a fine-toothed comb.

A.12. 10 to 11

- fixed yang validation issues found by IETFYANGPageCompilation. note: these issues were NOT found by pyang --ietf or by the submission-time validator...
- fixed a typo in the yang module, someone the config false statement was removed.

A.13. 11 to 12

- fixed typo that prevented Appendix B from loading the examples correctly.
- fixed more yang validation issues found by IETFYANGPageCompilation. note: again, these issues were NOT found by pyang --ietf or by the submission-time validator...
- updated a few of the notification enumerations to be more consistent with the other enumerations (following the warning/ error pattern).
- updated the information-type artifact to state how it’s encoded, matching the language that was in Appendix B.

A.14. 12 to 13

- defined a standalone artifact to encode the old information-type into a PKCS#7 structure.
- standalone information artifact hardcodes JSON encoding (to match the voucher draft).

- combined the information and signature PKCS#7 structures into a single PKCS#7 structure.
- moved the certificate-revocations into the owner-certificate's PKCS#7 structure.
- eliminated support for voucher-revocations, to reflect the voucher-draft's switch from revocations to renewals.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Mikael Abrahamsson
T-Systems

EMail: "mikael.abrahamsson@t-systems.se

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

M. Jethanandani
Cisco Systems, Inc

March 13, 2017

Accounting in NETCONF and RESTCONF
draft-mahesh-netconf-accounting-01

Abstract

This document defines an accounting record for NETCONF and RESTCONF.

Requirements Language

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].

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Terminology 2
2. Accounting Record 3
3. Data Model Definitions 3
3.1. Data Organization 3
3.2. YANG Module 4
4. IANA Considerations 8
5. Security Considerations 9
6. Acknowledgements 9
7. Normative References 9
Author's Address 10

1. Introduction

NETCONF [RFC6241] and RESTCONF [RFC8040] protocol operations are authenticated and authorized as part of the Authentication, Authorization and Accounting (AAA) framework. An accounting record needs to be created as part of the same framework for each of these operations to satisfy the accounting part of AAA. Having an accounting record that is consistent across vendors allows for the operator to compare operations across devices from different vendors. This document defines such a record and a corresponding YANG data model (ietf-netconf-am.yang).

The rest of this document will use NETCONF to imply both NETCONF and RESTCONF, but where applicable will call out each protocol specifically.

1.1. Terminology

The following terms are defined in NETCONF [RFC6241] and are not redefined here:

- client
- server
- session
- user

2. Accounting Record

An accounting record for NETCONF consists of the following fields.

acct-code	date-time	src-ip	session-id	task-id	user	groups	path	action	rule	status

where:

acct-code: START indicates a start of a new record, NONE a continuation, and STOP the end of the record.

date-time: The date and time when the operation was performed (UTC Timezone)

src-ip: The source IP address that was used to request the operation

session-id: The session-id in case of NETCONF and would be blank in case of RESTCONF

task-id: Used to track a accounting record in case it needs to split for uploading or storing. The id is a monotonically increasing number assigned by the server.

user: The NETCONF user that requesed this operation.

groups: The group the user belongs to.

path: The path in the NACM rule on which the operations is being performed

action: The action in the NACM rule

rule: The rule in the NACM that was used to authorize the action.

status: Whether the operations was permitted or denied.

3. Data Model Definitions

3.1. Data Organization

The following diagram highlights the contents and structure of the Accounting YANG module.

```
   module: ietf-netconf-am
      +--ro nam
         +--ro accounting-record* [task-id]
            +--ro task-id       uint32
            +--ro session-id?   nc:session-id-type
            +--ro acct-code     enumeration
            +--ro date-time     yang:date-and-time
            +--ro src-ip        inet:ip-address
            +--ro group         nacm:group-name-type
            +--ro user?         nacm:user-name-type
            +--ro path          nacm:node-instance-identifier
            +--ro action        nacm:access-operations-type
            +--ro rule?         string
            +--ro status?       nacm:action-type
```

3.2. YANG Module

The following YANG module specifies the normative NETCONF content that MUST be supported by the server.

The "ietf-netconf-am" YANG module imports typedefs from YANG-TYPES [RFC6991], from NETCONF [RFC6241] and from NACM [RFC6536].

<CODE BEGINS> file "ietf-netconf-am@2017-03-13.yang"

```
module ietf-netconf-am {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-am";
    prefix "nam";

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

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

    import ietf-netconf {
        prefix nc;
        //reference
        //    "RFC 6241: NETCONF Protocol";
    }
```

```
    import ietf-netconf-acm {
        prefix nacm;
        //reference
        //    "RFC 6536: NACM";
    }

    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:   Mahesh Jethanandani
                   <mailto:mjethanandani@gmail.com>";


    description
        "This module defines an accounting record for NETCONF operations
         performed on the server. If these operations are authorized
         using rules defined by NACM [RFC6536], then that information is
         also captured by this module.

         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 "2017-03-13" {
         description
             "Initial version";
         reference
             "RFC XXXX: NETCONF and RESTCONF Accounting";
     }
```

```
     /*
      * Data definition statements.
      */

     container nam {
         config false;
         description
             "Parameters for NETCONF Accounting Model.";

         list accounting-record {
             key "task-id";
             description
                 "A list of accounting records generated by the server";

             leaf task-id {
                 type uint32;
                 description
                     "The task-id is a monotonically increasing number
                      assigned by the server to capture a single
                      transaction.";
             }

             leaf session-id {
                 type nc:session-id-type;
                 description
                     "If this operation happened over NETCONF, this
                      field captures the NETCONF session-id. In case
                      of RESTCONF this field can be left blank.";
             }

             leaf acct-code {
                 type enumeration {
                     enum start {
                         description
                             "Start of an accounting record";
                     }
                     enum stop {
                         description
                             "Indicates the end of an accounting
                              record";
                     }
                     enum none {
                         description
                             "Indicates a single payload or a
                              continuation of an accounting record.";
                     }
                 }
                 mandatory true;
```

```
                description
                    "Some of the AAA server place a limit on the size
                     of the payload that can be transmitted at any
                     particular time.

                     This field indicates what constitues a complete
                     accounting record by setting up the boundaries. If
                     the accounting record fits within the payload
                     boundary the field should be set to none.";
            }

            leaf date-time {
                type yang:date-and-time;
                mandatory true;
                description
                    "The date and time when the operation was
                     requested.";
            }

            leaf src-ip {
                type inet:ip-address;
                mandatory true;
                description
                    "The source IP address where the request was made
                     from.";
            }

            leaf group {
                type nacm:group-name-type;
                mandatory true;
                description
                    "The name of the group that the user who requested
                     the operation belongs to.";
            }

            leaf user {
                type nacm:user-name-type;
                description
                    "The user within the group that is requesting this
                     operation.";
            }

            leaf path {
                type nacm:node-instance-identifier;
                mandatory true;
                description
                    "Data Node Instance Identifier associated with the
                     data node that the request is being made on.
```

```
                        Instance identifiers start with the top-level
                        data node, and a complete identifier is required
                        for this value.";
                }

                leaf action {
                    type nacm:access-operations-type;
                    mandatory true;
                    description
                        "The type of NETCONF operation being requested.";
                }

                leaf rule {
                    type string {
                        length "1..max";
                    }
                    description
                        "The name assigned to the rule that was used to
                         authorize the action, if authorization was
                         enabled.";
                }

                leaf status {
                    type nacm:action-type;
                    description
                        "Action taken by the server when the above
                         mentioned rule matched, if authorization was
                         enable.";
                }
            }
        }
}
```

<CODE ENDS>

4. IANA Considerations

This document makes two requests of IANA.

The first request is to register one URI in "The IETF XML Registry". Following the format in The IETF XML Registry [RFC3688], the following needs to be registered.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-am

Registrant Contact: The IESG

XML: N/A, the requested URI is an XML namespace

The second request is to register one module in the "YANG Module Names" registry. Following the format in YANG [RFC7950], the following needs to be registered.

Name: ietf-netconf-am

Namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-am

Prefix: nam

Reference: RFC XXXX

Note to RFC Editor - Please replace XXXX with the RFC id assigned to this draft.

5. Security Considerations

The YANG module defined in this memo is designed to be accessed using the NETCONF protocol. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH.

Most of the data nodes defined in this YANG module are readonly, i.e. config false, and are therefore not vulnerable in network environments. However, they might contain data that might be sensitive and should be protected with the right NACM [RFC6536] rules.

6. Acknowledgements

7. 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>.

[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>.

[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>.

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

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

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

Author’s Address

Mahesh Jethanandani
Cisco Systems, Inc
170 West Tasman Drive
San Jose, CA 95070
USA

Email: mjethanandani@gmail.com

NETMOD
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2017

E. Voit
Cisco Systems
A. Bierman
YumaWorks
A. Clemm
Huawei
T. Jenkins
Cisco Systems

February 21, 2017

YANG Notification Headers and Bundles
draft-voit-netmod-yang-notifications2-00

Abstract

There are useful capabilities not available with existing YANG notifications as described in Section 7.16 of [RFC7950]. These include:

1. what are the set of transport agnostic header objects which might be usefully placed within YANG notifications.

2. how might a set of YANG notifications be bundled into a single transport message.

3. how do you query the originator of a notification to troubleshoot the bundling process

This specification provides technologies enabling these three capabilities.

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 August 25, 2017.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Header Objects 3
4. Headers added to an RFC7950 Notification 4
5. Bundled Notifications 5
6. Querying an Object Model 7
7. Data Model 9
8. Security Considerations 19
9. References 20
9.1. Normative References 20
9.2. Informative References 20
Appendix A. Issues being worked 20
Authors' Addresses 21

1. Introduction

Mechanisms to support subscription to event notifications and yang datastore push are being defined in [sn] and [yang-push]. Work on those documents has shown that additional capabilities in YANG notifications would be helpful. Three of these capabilities include:

1. what are the set of transport agnostic header objects which might be usefully placed within YANG notifications.

2. how might a set of YANG notifications be bundled into a single transport message.

3. how do you query the originator of a notification to troubleshoot the bundling process.

As none of these three capabilities are specific to subscriptions, it would be good to define them in a transport protocol agnostic way.

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].

Definitions of Notification, Event, Event Notification, Publisher, Receiver, and Subscription are defined in [sn].

3. Header Objects

There are a number of transport independent headers which should have common definition across applications. These include:

- record-type: what kind of information and have been assembled as part of this notification. (E.g., is it a YANG datastore update, an alarm, a syslog message, etc.)
- subscription-id: provides a reference into the reason the originator believed the receiver wishes to be notified of this specific information.
- record-severity: how important the originator feels this message to be.
- record-time: the time an event notification itself was recorded in the originating system.
- record-id: identifies an event notification on an originator.
- observation-domain-id: identifies the originator process which discovered and recorded the event notification. (note: look to reuse the domains set up with IPFIX.)
- notification-time: the time the message was packaged sent to the transport layer for delivery to the receiver.
- signature: allows an application to sign a message so that a receiver can verify the authenticity of the message.
- notification-id: identifies a message which includes one or more event records.

- previous-notification-id: the Notification id previously sent to a receiver. When used in conjunction with notification-id, this allow loss/duplication across previous messages to be discovered.

- message-generator-id: process which created the message notification. Allows identification of different line cards sending the notification messages. Used in conjunction with previous-notification-id, can help find drops and duplication when notifications are coming from multiple sources on a device. The logic is simple: if there is a message-generator-id in the header, then the previous-notification-id should been the notification-id the last time the message-generator-id was sent.

4. Headers added to an RFC7950 Notification

With the headers defined, they may now be applied to extend an RFC-7950 notification. This section provides examples of this.

The first thing which is done is to encapsulate these header fields within their own subtree in the notification message so that these objects can easily be decoupled, processed, and removed from any notification record payload.

It is useful to sequence these objects so that processing by the receiver is as efficient as possible, allowing the discarding of uninteresting notifications as quickly as possible. One record priority encoding would include the objects presented in the sequence above to help minimize event record processing CPU cycles. (Need to add more here, and acknowledge that different payloads and systems might benefit from alternative sequencing.)

```
    +---n notification-message
       +--ro notification-message-header
       |  +--ro record-time
       |  +--ro record-type?
       |  +--ro record-id?
       |  +--ro record-severity?
       |  +--ro observation-domain-id?
       |  +--ro subscription-id?
       |  +--ro notification-time?
       |  +--ro notification-id?
       |  +--ro previous-notification-id?
       |  +--ro signature?
       |  +--ro message-generator-id?
       +--ro receiver-record-contents?
```

An actual instance of a notification might look like:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netmod:notification:2.0">
  <notification-message-header>
      <record-time>
          2017-02-14T00:00:02Z
      </record-time>
      <record-type>
          yang-patch
      </record-type>
      <subscription-identifier>
          823472
      </subscription-identifier>
      <notification-time>
          2017-02-14T00:00:05Z
      </notification-time>
      <notification-identifier>
          456
      </notification-identifier>
      <previous-notification-identifier>
          567
      </previous-notification-identifier>
      <signature>
          lKIo8s03fd23.....
      </signature>
  </notification-message-header>
  <datastore-changes>
      ...(yang patch here)...
  </datastore-changes>
</notification>
```

5. Bundled Notifications

In many implementations, it may be inefficient to transport every notification independently. Instead, scale and processing speed can be improved by placing multiple notifications into one transportable bundle.

When this is done, one additional of a header field becomes valuable. This is the "record-count" which would tally the quantity of records which make up the contents of the bundle.

The format of a bundle would look as below. When compared to the unbundled notification, note that the headers have been split so that one set of headers associated with the notification occur once at the beginning of the message, and additional record specific headers which occur before individual records.

```
      +---n bundled-notification-message
         +--ro notification-message-header
         |  +--ro notification-time
         |  +--ro notification-id?
         |  +--ro previous-notification-id?
         |  +--ro signature?
         |  +--ro message-generator-id?
         |  +--ro record-count?
         +--ro notification-records*
            +--ro notification-record-header
            |  +--ro record-time
            |  +--ro record-type?
            |  +--ro record-id?
            |  +--ro record-severity?
            |  +--ro observation-domain-id?
            |  +--ro subscription-id?
            +--ro receiver-record-contents?
```

An actual instance of a bundled notification might look like:

```
     <notification
       xmlns="urn:ietf:params:xml:ns:netmod:notification:2.0">
       <bundled-notification-message-header>
           <notification-time>
               2017-02-14T00:00:05Z
           </notification-time>
           <notification-identifier>
               456
           </notification-identifier>
           <previous-notification-identifier>
               567
           </previous-notification-identifier>
           <signature>
               lKIo8s03fd23...
           </signature>
           <record-count>
               2
           </record-count>
       </bundled-notification-message-header>
       <notification-record>
           <notification-record-header>
               <record-time>
                   2017-02-14T00:00:02Z
               </record-time>
               <record-type>
                   yang-patch
               </record-type>
               <subscription-identifier>
                   823472
               </subscription-identifier>
           </notification-record-header>
           <notification
              xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
              <datastore-changes>
                ...(yang patch here)...
              </datastore-changes>
           </notification>
       </notification-record>
       <notification-record>
               ...(record #2)...
       </notification-record>
     </notification>
```

6. Querying an Object Model

It is possible that an administrator would like to examine the contents of notifications via random access using a YANG model. There could be several values in such random access. These include:

- ability for applications to determine what message bundles were used to transport specific records.
- ability for applications to check which receivers have been sent an event notification.
- ability for applications to determine the time delta between event identification and transport.
- ability to reconstruct message passing during troubleshooting.
- ability to extract messages and records to evaluate whether the security filters have been properly applied.
- ability to compare the payloads of the same notification message sent to different receivers (again to evaluate the impact of the security filtering).

If such random access is needed, the YANG model structure below would enable random access to the information.

```
    +--ro notification-records
    |  +--ro notification-record* [record-id]
    |     +--ro record-time                    yang:date-and-time
    |     +--ro record-type              notification-record-format-type
    |     +--ro record-id                      uint32
    |     +--ro record-severity?               string
    |     +--ro observation-domain-id?         string
    |     +--ro notification-record-contents
    |     +--ro subscription-id*               subscription-ref
    +--ro notification-messages
    |  +--ro notification-message* [notification-id]
    |     +--ro notification-id                 uint32
    |     +--ro signature?                      string
    |     +--ro message-generator-id?           string
    |     +--ro notification-record             notification-record-ref
    |     +--ro receiver-notification-messages
    |        +--ro receiver-notification-message*
    |           +--ro receiver?                  receiver-ref
    |           +--ro notification-time          yang:date-and-time
    |           +--ro previous-notification-id?  uint32
    |           +--ro receiver-record-contents
    +--ro bundled-notification-messages
       +--ro bundled-notification-message* [notification-id]
          +--ro notification-id                   uint32
          +--ro signature?                        string
          +--ro message-generator-id?             string
          +--ro included-notification-records
          |  +--ro included-notification-record*
          |     +--ro notification-record?   notification-record-ref
          +--ro receiver-notification-messages
             +--ro receiver-notification-message*
                +--ro receiver?                      receiver-ref
                +--ro notification-time              yang:date-and-time
                +--ro previous-notification-id?      uint32
                +--ro record-count?                  uint16
                +--ro included-notification-records
                   +--ro notification-record*
                      +--ro receiver-record-contents
```

If such random access is not seen as needed, the model above should be discarded. This will also simplify the YANG model is the following section.

7. Data Model

```
 <CODE BEGINS> file "ietf-yang-notifications2.yang"
 module ietf-yang-notifications2 {
   yang-version 1.1;
```

```
   namespace "urn:ietf:params:xml:ns:yang:ietf-yang-notifications2";
   prefix yn2;

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

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

      WG Chair: Lou Berger
                <mailto:lberger@labn.net>

      WG Chair: Kent Watsen
                <mailto:kwatsen@juniper.net>

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

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

      Editor:   Tim Jenkins
                <mailto:timjenki@cisco.com>

      Editor:   Andy Bierman
                <mailto:andy@yumaworks.com>";


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

   revision 2017-02-23 {
     description
       "This module includes several definitions for two new yang
       notification message formats:
       (a) a message format including the definitions for common header
           information prior to notification payload.
       (b) a message format allowing the bundling of multiple
           notifications within it

       It also includes data nodes for querying related information such
```

```
       as:
          - ability to see contents of notifications both before and
            after any NACM filtering has been applied.
          - ability to see which message numbers have been sent to which
          receivers.";

     reference
       "draft-voit-netmod-yang-notifications2-00";
   }

   /*
    * IDENTITIES
    */

    /* Identities for notification record types */

    identity notification-record-format {
     description
       "Base identity to represent a different formats for notification
       records.";
   }

   identity system-event {
     base notification-record-format;
     description
       "System XML event";
   }

   identity yang-datastore {
     base notification-record-format;
     description
       "yang tree info";
   }

   identity yang-patch {
     base notification-record-format;
     description
       "yang patch record";
   }

   identity syslog-entry {
     base notification-record-format;
     description
       "Entry into syslog (figure linkage to existing draft.";
   }

   identity alarm {
     base notification-record-format;
```

```
    description
      "Alarm (perhaps link draft-sharma-netmod-fault-model-01 for more
      info)";
  }

  /*
   * TYPEDEFs
   */

  typedef notification-record-ref {
    type leafref {
      path "/notification-records/notification-record/record-id";
    }
    description
      "This type is used to reference a notification record (a.k.a.,
      event).";
  }

  typedef receiver-ref {
    type leafref {
      path "/sn:subscriptions/sn:subscription/sn:receivers/"+
      "sn:receiver/sn:address";
    }
    description
      "This type is used to reference a receiver.";
  }

  typedef subscription-ref {
    type leafref {
      path "/sn:subscriptions/sn:subscription/sn:identifier";
    }
    description
      "This type is used to reference a receiver.";
  }

  typedef notification-record-format-type {
    type identityref {
      base notification-record-format;
    }
    description
      "Type of notification record";
  }

  /*
   * GROUPINGS
   */

  grouping notification-message-header {
```

```
    description
      "Header information included with a notification.";
    leaf notification-id {
      type uint32;
      description
        "unique id for a notification which may go to one or many
        receivers.";
    }
    leaf signature {
      type string;
      description
        "Any originator signing of the contents of a notification
        message.  This can be useful for originating applications to
        verify record contents even when shipping over unsecure
        transport.";
    }
    leaf message-generator-id {
      type string;
      description
        "Software entity which created the notification message (e.g.,
        linecard 1).";
    }
  }

  grouping notification-message-receiver-header {
    description
      "Header information included with a notification which is
      specific to a receiver.";
    leaf notification-time {
      type yang:date-and-time;
      description
        "time the notification was generated prior to being sent to
        transport.";
    }
    leaf previous-notification-id {
      type uint32;
      description
        "Notification id previously sent by publisher to a specific
        receiver (allows detection of loss/duplication).";
    }
  }

  grouping notification-record-header {
    description
      "Common informational objects which might help a receiver
      interpret the meaning, details, and importance of an event
      notification.";
    leaf record-time {
```

```
      type yang:date-and-time;
      mandatory true;
      description
        "Time the system recognized the occurence of an event.";
    }
    leaf record-type {
      type notification-record-format-type;
      description
        "Describes the type of payload included.  This is turn allow
        the interpretation of the record contents.";
    }
    leaf record-id {
      type uint32;
      description
        "Identifier for the notification record.";
    }
    leaf record-severity {
      type string;
      description
        "System assigned severity.  (Likely we need to build/find an
        enumeration of common ones.)";
    }
    leaf observation-domain-id {
      type string;
      description
        "Software entity which created the notification record (e.g.,
        process id).";
    }
  }

  grouping subscribed-notification-record-header {
    description
      "Header information included with a notification.";
    uses notification-record-header;
    leaf subscription-id {
      type uint32;
      description
        "Id of the subscription which led to the notification being
        generated.";
    }
  }

  /*
   * NOTIFICATIONS
   */

  notification notification-message {
    description
```

```
      "Notification message to a receiver containing only one event.";
    container notification-message-header {
      description
        "delineates header info from notification messages for easy
        parsing.";
        uses subscribed-notification-record-header;
        uses notification-message-header;
        uses notification-message-receiver-header;
    }
    anydata receiver-record-contents {
      description
        "Non-header info of what actually got sent to receiver after
        security filter.  (Note: Possible to have extra process
        encryption.)";
    }
  }

  notification bundled-notification-message {
    description
      "Notification message to a receiver containing many events,
      possibly relating to independent subscriptions.";
    container bundled-notification-message-header {
        description
            "Delineates header info from notification messages for easy
            parsing.";
        uses notification-message-header;
        uses notification-message-receiver-header {
          refine notification-time {
            mandatory true;
          }
        }
        leaf record-count {
            type uint16;
            description
                "Quantity of events in a bundled-notification-message
                for a specific receiver.  This value is per receiver in
                case an entire notification record is filtered out.";
        }
    }

    list notification-records {
      description
        "Set of messages within a notification to a receiver.";
      container notification-record-header {
        description
          "delineates header info from notification messages for easy
          parsing.";
        uses subscribed-notification-record-header;
```

```
      }
      anydata receiver-record-contents {
        description
          "Non-header info of what actually got sent to receiver after
          security filter.  (Note: Possible to have extra process
          encryption.)";

      }
    }
  }

  /*
   * DATA NODES
   */

  container notification-records {
    config false;
    description
      "Maintains instances of event notifications recorded by the
      system.";
    list notification-record {
      key "record-id";
      description
        "Specific instances of event notifications recorded by the
        system.";
      uses notification-record-header {
        refine record-id {
          mandatory true;
        }
        refine record-type {
          mandatory true;
        }
      }
      anydata notification-record-contents {
        mandatory true;
        description
          "Notification event contents independent of any receiver
          security filtering.";
      }
      leaf-list subscription-id {
        type subscription-ref;
        description
          "Instances of subscriptions which should receive or have
          received this event notification record.";
      }
    }
  }
  container notification-messages {
```

```
    config false;
    description
      "Contains a history of the notification messages which have been
      generated.";
    list notification-message {
      key "notification-id";
      description
        "Instances of notification messages generated with the intent
        of sending them to one or more receivers.";
      uses notification-message-header {
        refine notification-id {
          mandatory true;
        }
      }
      leaf notification-record {
        type notification-record-ref;
        mandatory true;
        description
          "Included notification. The record itself, or elements of
          this record might not be sent to any included receiver based
          on security permissions for that receiver.";
      }
      container receiver-notification-messages {
        description
          "Contains a history of messages targeted for a receiver.";
         list receiver-notification-message {
          description
            "Maintains instances of messages targeted for a receiver.";
          leaf receiver {
            type receiver-ref;
            description
              "Reference to the recipient targeted for this
              notification message.  (This also allows the unique
              identification of the subscription.)";
          }
          uses notification-message-receiver-header {
            refine notification-time {
              mandatory true;
            }
          }
          anydata receiver-record-contents {
            mandatory true;
            description
              "The specific security filtered contents of one record
              going to a receiver.";
          }
        }
      }
```

```
    }
  }
  container bundled-notification-messages {
    config false;
    description
      "Contains a history of bundled notification messages which have
      been generated.";
    list bundled-notification-message {
      key "notification-id";
      min-elements 1;
      description
        "Maintains instances of a bundled notification messages
        generated with the intent of sending them to one or more
        receivers.";
      uses notification-message-header{
        refine notification-id {
          mandatory true;
        }
      }
      container included-notification-records {
          description
          "Contains specific records included in the bundle.";
        list included-notification-record {
          description
            "A specific instance of record included in a bundle.";
          leaf notification-record {
            type notification-record-ref;
            description
              "Included notification within the bundle.  Full records
              or elements of this record might not be sent to any
              included receiver based on security permissions for that
              receiver.";
          }
        }
      }
      container receiver-notification-messages {
          description
          "Contains instances of messages generated for a specific
          receiver.";
         list receiver-notification-message {
          description
            "Maintains instances of bundled messages targeted for a
            receiver.";
           leaf receiver {
            type receiver-ref;
            description
              "Reference to the recipient targeted for this bundled
              notification message.  (As a receiver is unique to a
```

```
                subscription, this also identifies the subscription
                explicitly.  If something other than receiver is used, a
                method to identify the subscription is also needed as it
                can't automatically be derived from the notification
                record.";
            }
            uses notification-message-receiver-header {
              refine notification-time {
                mandatory true;
              }
            }
            leaf record-count {
              type uint16;
                description
                "Number of records actually sent to a receiver after
                considering the application of NACM policies on the
                notification records.";
            }
            container included-notification-records {
              description
                "Contains the records sent to a receiver within a
                specific notification message.";
              list notification-record {
                description
                  "Maintains instances of records sent to a receiver.";
                anydata receiver-record-contents {
                  mandatory true;
                    description
                    "The specific security filtered contents of one
                    record going to a receiver.";
                }
              }
            }
          }
        }
      }
    }
  }
}
<CODE ENDS>
```

8. Security Considerations

to be populated

9. References

9.1. 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>.

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

9.2. Informative References

[sn] Voit, E., Clemm, A., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to Event Notifications", February 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>.

[yang-push] Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", February 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

Appendix A. Issues being worked

(To be removed by RFC editor prior to publication)

We need to define the ways to invoke and configure the capability within an originating device. This includes defining what header types are selected.

Should we allow multiple subscriptions to be associated with an update record via a leaf-list?

Should the subscription id in a notification actually be a leafref?

We need to do a lot more to discuss transport efficiency implications.

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Tim Jenkins
Cisco Systems

Email: timjenki@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

Z. Wang
G. Zheng
Huawei Technologies

March 13, 2017

Network Configuration Protocol (NETCONF) Proxy
draft-wangzheng-netconf-proxy-00

Abstract

This document presents Network Configuration Protocol (NETCONF) Proxy through which NETCONF requests can be forwarded to a target host. It would be useful when a client does not have direct network access to a target host.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Motivation 2
1.2. Requirements Terminology 3
2. Solution Overview 3
3. The NETCONF Client 5
4. The Proxy 6
5. The Target 7
6. New attribute: target-id 8
7. YANG DATA MODEL 8
7.1. Overview 8
7.2. YANG Module 9
8. Security Considerations 11
9. IANA Considerations 11
10. Normative References 11
Authors' Addresses 11

1. Introduction

This document proposes a NETCONF Proxy mechanism. The mechanism extends the NETCONF protocol [RFC6241] which provides a standard way to set up NETCONF session. At its core, the mechanism defined here introduces a set of new operations which allow a client to forward NETCONF requests to a target host through an intermediary NETCONF proxy server, especially in case where client would otherwise not have direct network access to a target host. The document also includes YANG data model which extend the model and RPCs defined within [RFC6241].

1.1. Motivation

NETCONF provide a RPC-based mechanism to facilitate communication between a client and a server. The client can be a script or application typically running as part of a network manager. The server is typically a network device [RFC6241]. However, the network manager may not have direct network access to the target network devices. For example, some target network devices may locate in a network with private addresses behind a NAT device or a firewall. Thus, network manager cannot direct communicate with these target devices.

NETCONF Call Home [RFC8071] provides a mechanism that allows NETCONF Servers to initiate a connection with a NETCONF client, reversing the normal direction of NETCONF session setup. This allows a NETCONF Server, e.g. a networking device that needs to be managed, to reach out to a NETCONF Client, e.g. an Operations Support System of an SDN controller, in order to be managed. By reversing the direction in

which NETCONF sessions are normally set up, problems such as establishing connectivity with devices behind a firewall can be alleviated. However, NETCONF Call Home requires that the server knows its client by way of configuration or discovery. It does not address the scenarios as presented below:

1. In some NFV scenarios, VNF instances are running in a private network. To reduce the management resources (like IP resources, bandwidth, etc) of large-scale management activities, these VNF instances may not be assigned IP addresses. Then the element management system (EMS), which located in public network, cannot be aware of the addresses of VNF instances. Therefore, the element management system (EMS) is difficult to manage these VNF instances via NETCONF protocol.

2. And in some cloud network scenarios, the gateway network element (GNE) and non-gateway network elements (N-GNEs) communicate with each other using some private protocol. And these non-gateway network elements (N-GNEs) may not IP devices. Therefore, the cloud centre EMS (element management system) cannot be aware of the addresses of N-GENs. Thus, the element management system (EMS) is difficult to manage these N-GNE devices via NETCONF protocol.

To solve that problem, this document proposes a NETCONF Proxy mechanism. The proxy can acts as an intermediary between manager and target device, therefore the client can set up a NETCONF session to a target through a NETCONF Proxy.

The mechanism allows the client to subsequently direct NETCONF requests to the server, to receive responses, and to subscribe to notifications from the server. While the NETCONF Proxy can be used to traverse NAT boundaries, it should be noted that it does not apply NAT mappings for contents carried as part of the NETCONF payload; specifically, it does not substitute IP address information that is carried as part of data nodes.

1.2. Requirements 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].

2. Solution Overview

The diagram below illustrates how the client can set up a NETCONF session to a target through the NETCONF proxy.

This diagram makes the following points:

1. The client initiates the connection using the SSH/TLS transport protocol. When the NETCONF session is established, the client and proxy MUST send a <hello> element containing a list of that peer's capabilities. The proxy SHOULD send at least the "netconf" and "proxy" capabilities. And other rules of capabilities exchange described in section 8 of [RFC6241].

2. The client sends a <get> RPC to proxy to retrieve the "target-list" of the proxy.

3. The proxy responds with a <get-reply> RPC which containing "target-list" attributes. The "target-list" attributes includes

the target's information such as target-id, protocol, authentication, etc.

4. The client receive a <get-reply> RPC from the proxy, and looks up the value of "target-list". Then the client constructs a <hello> message according to the received "target-list". This <hello> massage SHOULD contain at least a "target-id" attribute. And then client sends this <hello> message to proxy and waits for a reply.

5. The proxy receives the <hello> message and checks the value of "target-id" attribute:

   If the target is not found, then an "invalid-target" error will be returned.

   If the target can be found, then the proxy initiates a connection to corresponding target. And then proxy forwards the <hello> message, which received from client, to corresponding target.

6. The target receives the <hello> message and then responds a <hello> message containing a list of capabilities. The rules of capabilities exchange described in section 8 of [RFC6241].

7. Now, client established a NETCONF session to a target through NETCONF Proxy. Subsequently, the client can direct NETCONF requests to the target, to receive responses, and to subscribe to notifications from the target.

3. The NETCONF Client

The term "client" is defined in [RFC6241], Section 1.1 "client". In the context of network management, the NETCONF client might be a network management system for example a EMS (element management system).

The client operation describes as follows:

1. The client initiates a connection to proxy using the SSH/TLS transport protocol [RFC6242]. How to establish an SSH/TLS transport connection is described in [RFC6242]

2. When the NETCONF session is established, the client sends a <hello> message to proxy, then waits for a reply. This <hello> message contains a list of client's capabilities.

3. After capabilities exchange, the client sends a <get> RPC to proxy to retrieve the "target-list" of the proxy, then waits for a reply.

4. The client receive a <get-reply> RPC from the proxy, looks up the value of "target-list", and then constructs a <hello> message according to the received "target-list". This <hello> massage SHOULD contain at least a "target-id" attribute. For example, if the client received "target-list" containing "target-id=A1", then the client constructs <hello> message:

```
C: <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
C:   <target-id>A1</target-id>
C:   <capability>foo<capability>
C: </hello>
```

And then client sends this <hello> message to proxy and waits for a reply.

5. If the reply presents the "capabilities" of target, the proxy connection is established. Subsequently, the client can direct NETCONF requests to the target, to receive responses, and to subscribe to notifications from the target.

6. If the reply contains the "invalid-target" error, the process turn to step (4) or aborts.

7. Otherwise, the client interprets the error and aborts.

4. The Proxy

The Proxy should ensure that requests given by client for a particular target device should reach the target device and the operations should be executed on that target device and the response should be given back to the client.

The proxy operation describes as follows:

1. When the NETCONF session is established, the proxy sends a <hello> element containing a list of proxy’s capabilities. The proxy SHOULD send at least the "netconf" and "proxy" capabilities. And other rules of capabilities exchange described in section 8 of [RFC6242].

2. The proxy receives the <get> RPC and then responds with a <get-reply> RPC which containing "target-list" attributes. The "target-list" attributes SHOULD includes the target's information such as target-id, protocol, authentication, etc. The following example shows a "target-list":

```
<target-list>
 <target-id>A1</target-id>
 <protocol>protocol-foo</protocol>
 <authentication>authentication-foo</authentication>
</target-list>
```

3. The proxy receives the <hello> message and checks the value of "target-id" attribute:

   If the target is not found in target-list, then an "invalid-target" error will be returned.

   If the target can be found in target-list, the proxy checks the corresponding "protocol" and "authentication" of the "target-id". And then, the proxy uses corresponding protocol to establish a connection to the target. After session established, the proxy forwards the <hello> message, which received from client, to corresponding target.

4. If the reply presents the "capabilities" of target, the proxy connection is established. Subsequently, the proxy transports the messages received from the client to the target and vice versa.

5. The Target

The term "target" is equivalent to the term "server" which is defined in [RFC6242], Section 1.1 "server". In the context of network management, the target is typically a network device.

The target operations describes as follows:

1. If the connection between the proxy and the target established. And target receives the <hello> message from the proxy, and then responds a <hello> message containing a list of capabilities. The rules of capabilities exchange described in section 8 of [RFC6242].

2. After client established a NETCONF session to a target through NETCONF Proxy. The client sends a series of one or more RPC request messages, which cause the server to respond with a corresponding series of RPC reply messages.

6. New attribute: target-id

A proxy can be used by a client to connect to several servers and to maintain multiple NETCONF sessions. A client may use the proxy even to maintain multiple NETCONF sessions with the same NETCONF server. When issuing a NETCONF request, a client must therefore differentiate between NETCONF sessions. To solve this problem, a new attribute "target-id" is defined. This attribute allow the proxy to forward RPC to corresponding target.

For example:

The following <rpc> element invokes the NETCONF <get> method and includes the "target-id" attribute:

```
<rpc message-id="101"
     target-id="A1"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get/>
</rpc>

<rpc-reply message-id="101"
     target-id="A1"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <!-- contents here... -->
  </data>
</rpc-reply>
```

7. YANG DATA MODEL

7.1. Overview

The YANG data model for NETCONF Proxy is depicted in the following figure. Following Yang tree convention in the depiction, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. A "+" at the end of a line indicates that the line is to be concatenated with the subsequent line.

```
module: ietf-netconf-proxy
    +--rw proxy {proxy}?
       +--rw proxy-name?    string
       +--rw target-list* [target-id]
          +--rw target-id         string
          +--rw protocol?         string
          +--rw authentication?   string
```

7.2. YANG Module

```
<CODE BEGINS> file "ietf-netconf-proxy@2017-03-09.yang"
module ietf-netconf-proxy {

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-proxy";

  prefix np;

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

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

     WG Chair: Mehmet Ersue
               mehmet.ersue@nsn.com

     Editor:   zitao wang
                   wangzitao@huawei.com";

  description
    "NETCONF Protocol Data Types and Protocol Operations.

     Copyright (c) 2011 IETF Trust and the persons identified as
     the document authors.  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 YANG module describe how to define a netconf proxy";

  revision 2017-03-09 {
    description
```

```
        "Initial revision";
      reference
        "draft-wang-netconf-proxy";
    }

    feature proxy {
      description
        "Netconf proxy";
    }

    container proxy {
     if-feature proxy;
     leaf proxy-name{
      type string;
      description
          "Proxy name";
     }
     list target-list {
      key "target-id";
          leaf target-id{
           type string;
           description
           "Target ID";
          }
          leaf protocol {
           type string;
           description
           "Support protocols";
          }
          leaf authentication {
           type string;
           description
           "Authentication";
          }
          description
          "List for target information";
     }
     description
     "Container for NETCONF Proxy";
    }

  }
  <CODE ENDS>
```

8. Security Considerations

The security considerations described in [RFC6242] and [RFC7589], and by extension [RFC4253], [RFC5246] apply here as well.

9. IANA Considerations

TBD

10. 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>.

[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>.

[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>.

[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>.

[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>.

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

Authors' Addresses

Zitao Wang
Huawei Technologies
101 Software Avenue, Yuhua District
Nanjing
China

EMail: wangzitao@huawei.com

Guangying Zheng
Huawei Technologies
101 Software Avenue, Yuhua District
Nanjing
China

EMail: zhengguangying@huawei.com